Republic of Iraq
Ministry of Higher Education
and Scientific Research
Al-Nahrain University
College of Science

IMPLEMENTATION OF LINK CHECKER FOR ANALYZING WEB SITE

A Thesis Submitted to the College of Science, Al-Nahrain University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

By
EHSAN QAHTAN AHMED
(B.Sc. 2002)

Supervised By

DR. ABDUL MONEM S. RAHMA **DR. JAMAL F. TAWFEQ**

October 2009 **Shawal 1430**

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

وَيَسئَلوُنَكَ عَنِ ٱلرُّوحِ قُلِ ٱلرُّوحُ مِن أَمرِ

رَبي وَمَاأُوتِيتُم مِّنَ ٱلعِلمِ إلاَّ قَليلاً ﴿٨٥﴾

صدق الله العظيم

الاسراء

SUPERVISOR CERTIFCATION

We certify this thesis was prepared under our supervision at the Department of Computer Science/College of Science/ Al-Nahrain University, By **Ehsan Qahtan Ahmed** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Signature:
Name: **Dr. Abdul Monem S. Rahma**
Title: **Professor**
Date: **/ / 2009**

Signature:
Name: **Dr. Jamal F. Tawfeq**
Title: **Lecturer**
Date: **/ / 2009**

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:
Name : **Dr. Taha S. Bashaga**
Title : **Head of the Department of Computer Science, Al-Nahrain University.**
Date : **/ / 2009**

CERTIFICATION OF THE EXAMINATION COMMITTEE

We chairman and members of the examination committee certify that we have studies this thesis "**Implementation of Link Checker for Analyzing Web Site**" presented by the student **Ehsan Qahtan Ahmed** and examined her in its contents and that we have found it worthy to be accepted for the degree of Master of Science in Computer Science with **good degree**.

Signature:
Name: **Dr. Murtadha M. Hamad**
Title: **Assistant Professor**
Date: **/ / 2010**
(Chairman)

Signature:
Name: **Dr. Baraa A. Atia**
Title: **Assistant Professor**
Date: **/ / 2010**
(Member)

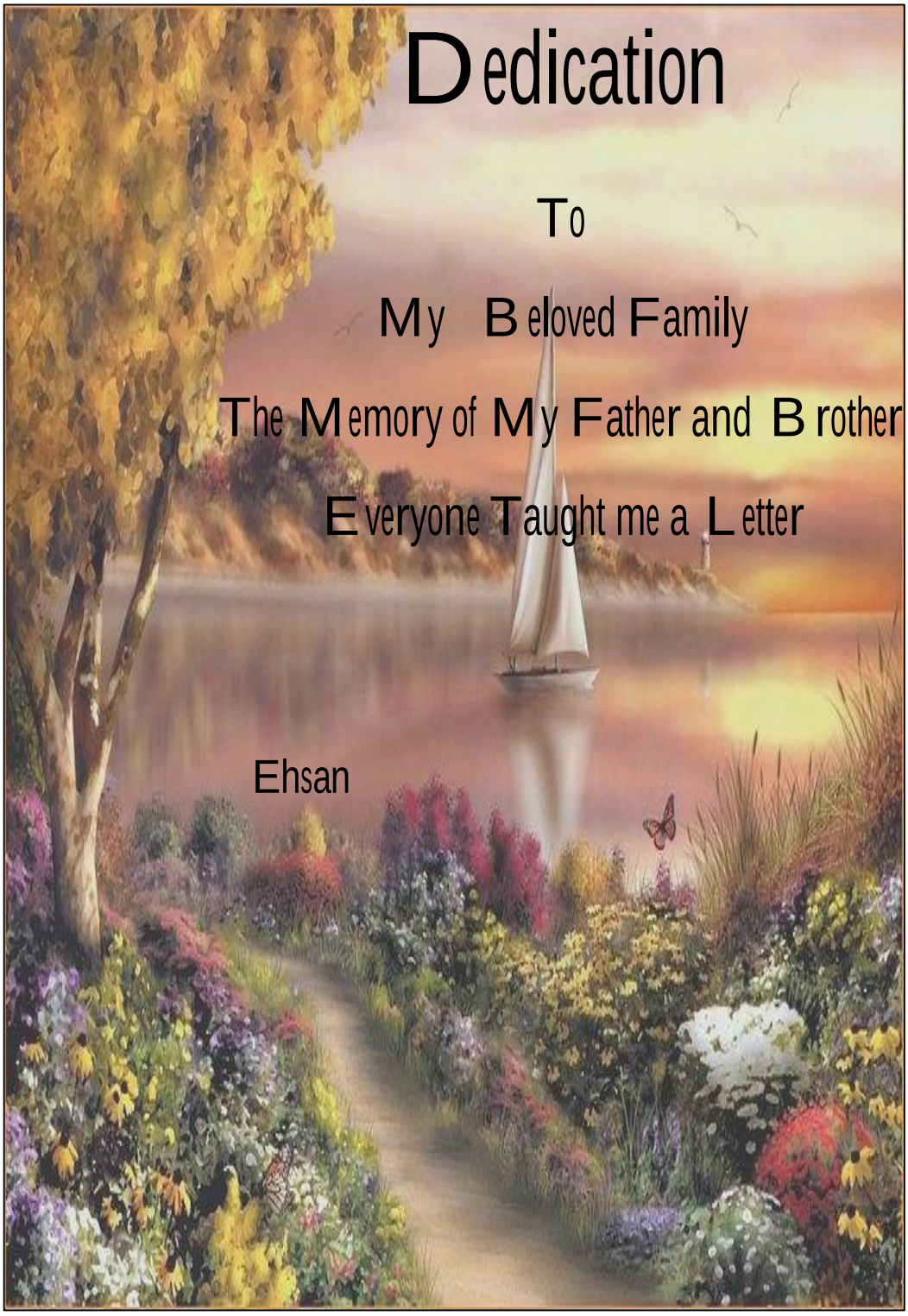
Signature:
Name: **Dr. Osama A. Awad**
Title: **Lecturer**
Date: **/ / 2010**
(Member)

Signature:
Name: **Dr. Abdul Monem S. Rahma**
Title: **Professor**
Date: **/ / 2010**
(Supervisor)

Signature:
Name: **Dr. Jamal F. Tawfeq**
Title: **Lecturer**
Date: **/ / 2010**
(Supervisor)

Approved by the Dean of the Collage of Science, Al-Nahrain University.

Signature:
Name: **Dr. LAITH ABDUL AZIZ AL-ANI**
Title: **Assist. Prof**.
Date: **/ / 2010**
(Dean of Collage of Science)


Dedication
To
My Beloved Family
The Memory of My Father and Brother
Everyone Taught me a Letter
Ehsan

Acknowledgment

First, I would like to thank God for all the blessings that have given us.

Second, I would like to express my sincere appreciation to my supervisors Dr. Abul Monem S. Rahma and Dr. Jamal F. Tawfeq for their valuable guidance, supervision and untiring efforts during the course of this work.

Grateful thanks for the Head of Department of Computer Science Dr. Taha S. Bashaga, staff, employees and everyone who teaches me.

Finally, my very special thanks to my family, the faithful friend who teaches me the alphabetic of the life and my lovely friends for continuous supports and encouragement during the period of my studies.

Ehsan

ABSTRACT

Broken links in a Web site is common problem on the Internet today. It inconvenience visitors, inhibits proper navigation of a site, prohibit access to Web site content and reduce the productivity of Web professionals.

This thesis aim to design and implement a software that test website links and then classified them to four lists according to link's type (relative-link list, fragment-link list, absolute-link list, image-link list), after classification process, the proposed link checker software check all links in each list to verify if link work correctly this means it is good link else it is a broken link.

The structure of proposed link checker software consists of three main modules. The first is called **crawler and extractor module**, the second is called **extractor image link module** and third is called c**hecker module** which consists of two sub-modules **text link checker and image link checker.**

The proposed link checker is implemented for HTML Web Sites. A Web Site for the computer science department at al Nahrain University is taken as a test bed.

The result obtained shows the applicability of the proposed link checker software to discover all the broken links in the Web Site. Also some statistical records are evaluated by the software, which is found useful for pre-Site evaluation and Web Sites developers.

The presented link checker can be used as a useful tool, for offline maintenance.

It is especially helpful to run as a final step before uploading the Web Site for production.

The programming tools used in proposed software are: Microsoft Visual Basic 6.0, Hypertext Markup Language and Windows operating system.

LIST OF ABBREVIATIONS

Abbreviation	Meaning
CERN	European laboratory for particle physics
DNS	Domain Name System
FRESS	File Retrieval and Editing System
FTP	File Transfer Protocol
HREF	Hypertext References
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers Standard Associative
KDE	Kernel Density Estimation
KMU	Knowledge Management System
Memex	Memory and Index System
MIT	Massachusetts Institute of Technology
NLS	On Line System
SGML	Standard Generalized Markup Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
Xerox	Xerox Corporation
XHTML	Extensible Hypertext Markup Language

TABLE OF CONTENTS

Chapter One: General Introduction

Chapter Two: Web and Hyperlinks

Chapter Three: Design of Proposed Link Checker Software

Chapter Four: Implementation of Proposed Link Checker Software

Chapter Five: Conclusions and Suggestions for Future Work

References

Chapter One

General Introduction

Chapter One
General Introduction

1.1 Introduction

One of most serious problems plaguing the World Wide Web today is that of broken links, which is a major annoyance to browsing users and also a cause of tarnished reputation and possible loss of opportunity for information providers. The root of the problem lies in the current Web architecture’s lack of support for referential integrity[Vir01].

1.2 World Wide Web's hyperlink

The World Wide Web (known as "WWW', "Web" or"W3") is the universe of network-accessible information, the embodiment of human knowledge.

The Web has a body of software, and a set of protocols and conventions. Through the use hypertext and multimedia techniques, the web is easy for anyone to roam, browse, etc.

The WWW is also a distributed hypermedia environment consisting of documents from around the world. The documents are linked using a system known as **Hypertext**, where elements of one document may be linked to specific elements of another document. The documents may locate on any computer connected to the internet. In this context, the world "document" is not limited to text but may include video, audio,

graphics, databases, and a host of other tools that can be accessed from any web browser [Dav00].

These documents are created with a special language called Hypertext Markup Language(HTML).This language allows the full use of the hypermedia including text, images, graphics, sounds and other types of multimedia[Sab02].

Hyperlinks allow user to weave together all HTML pages seamlessly for navigation from one page to the next. Without links, websites would be virtually worthless and very boring. The link also called hyperlink or web link, is basic hypertext construct. A link is a connection from one web source to another. Although a simple concept, the link has been one of the primary forces driving the success of the web [Jen99].

When look back to the history of hypertext and show different hypertext systems each using their own way to deal with navigation an orientation problem.

1. 1945 Vannevar bush proposed Memory and Index (Memex) in his article "As We May Think". Memex was designed as a mechanical hypertext device that used microfilm.
2. 1965 Ted Nelson introduced the Xanadu distributed system concept and coined the term hypertext. It took until 1999 to actually build Xanadu.
3. 1967 Andries van Dam developed the Hypertext Editing System at Brown University, followed by the introduction of File Retrieval and Editing System (FRESS) in 1968.

4. 1968 Doug Engelbart gave a demo of On Line System(NLS), the NLS, as an experimental tool to store specifications, plans, designs, programs, documentation, reports, etc.
5. 1975 A team at Carnegie Mellon University (CMU), headed by Robertson, developed the Zionist Occupation Government (ZOG) system, which later became Knowledge Management System (KMS).
6. 1978 A team at Massachusetts Institute of Technology System (MIT), headed by Andrew Lippman, developed the Aspen Movie Map, the first true example of a multimedia application including videodisk.
7. 1985 Janet Walker developed the Symbolic Document Examiner, claimed to be the first hypertext system used by "real" customers. The user-interface was kept as simple as possible.
8. 1985 Several other hypertext systems were announced, including NoteCards from Xerox and Intermedia from Brown University.
9. 1986 Office Workstation Ltd. (OWL) introduced Guide for the Macintosh, the rest widely available hypertext system, based on the Unix Guide system and developed by Peter Brown at the University of Kent.
10. 1987 Apple started delivering HyperCard for free with every Macintosh [Hon02].

1.3 The Broken Link's Problem

Most of internet's users faced the problem "file not found" in the internet which represent by error 404 this error message indicates that the server name is valid, but the web page could not be found at the specified location. Two things could be wrong.

1. The page might have moved or deleted. In this case, the user can try to find its new location. Strip off the filename and see whether the resulting URL takes him (her) to a related page. If it does, he (she) might be able to navigate back to the desired page. If that step fails, strip off the next directory in the Uniform Resource Locator (URL) and test that address. With a little luck one of the shorter URLs will produce a home page that he (she) can use to locate the correct page.
2. Error in the syntax of link. [Dav00]

Nonworking links can frustrate web site visitors, so keep web page in good operating condition by periodically verifying that all links still work correctly. It is not enough to know that a link was working when first created it.

After all, a link that works today might not work tomorrow. The page might be removed from the web or the page's author might rearrange some files of directories, rendering the old URL obsolete.

Ongoing maintenance is needed to ensure that hyperlinks remain operational next week, next month, and next year. This requirement is one of the hidden costs associated with posting pages on the web.

Although it is certainly fun to create new pages, most people find maintaining pages to be tedious, if one web page contains only 5 to 10 links, designer can check them manually and still keep his site in tip-top shape. Unfortunately, manual link checking soon becomes onerous, even for a small number of links. If the designer forgot this routine, his site will suffer and his users will feel neglected or annoyed and may never return. If his site is compelling and entertaining, his users may forgive him. Most of users can't count on such a loyal and understanding following, so it's important to keep all web page links operational and

current, to solve this big broken link's problem, the user need system that check all links of web site and find the broken links to correct them. [Chu98].

1.4 Literature Survey

There are some previous literature survey works on broken links to correct them:

1. Link Tek Corporation [Lin02], "Link Fixer Plus": This application specifically designed to fix broken links throughout the local copy of a Web site, totally automatically. By helping eliminate the tedium of manually finding and fixing broken links, Link Fixer Plus frees up time that Web professionals would otherwise spend manually repairing links throughout their Web sites. Additionally, by using Link Fixer Plus, Web professionals can more efficiently ensure the integrity of links in their Web sites, while simultaneously increasing their overall productivity.

2. Michael D. Larue [Mic02], "Link Controller": it is checking links have been broken for a period of time and then finding which documents they occur in.

3. Zubin Jamshed Dalal[Zub03], "Solving the Broken Link Problem in Walden's Paths": This thesis proposes an algorithm to extract representative keyphrases to locate exact copies of the original page. In the absence of an exact copy, a similar but separate algorithm is used to extract keyphrases that will help locating similar pages that can be substituted in place of the missing page. Both sets of keyphrases are stored as additions to

the page signature in the Walden's Paths Path Manager tool and can be used when the original page is removed from its current location on the Web.

4. Paulo Moura Guedes[Pau04], "KLinkStatus": is a link checker software for Kernel Density Estimation (KDE). It allows the user to search internal and external links in his (her) entire web site, just a single page and chooses the depth to search. The user can also check local files, ftp, fish, etc.

1.5 Aim of Thesis

This thesis aim to design and implement software that test website links and then classified them to four lists according to link's type (relative-link list, fragment-link list, absolute-link list, image-link list), after classification process, the proposed link checker system check all links in each list to verify if link work correctly this means it is good link else it is a broken link.

1.6 Thesis Outline

The reminder part of this thesis consists of four chapters, and they organized as follows:

Chapter Two reviews the relevant subjects surrounding the links, and illustrates about World Wide Web, also explain the component of website and the broken link's problem.

Chapter Three explain the design of the proposed link checker system in detailed.

Chapter Four explains the implementation of proposed system and how the user can use it.

Chapter Five gives a list of derived conclusions of the presented work. Also, in this chapter some proposals are expressed for the direction that the future work may take.

Chapter Two

Web and Hyperlinks

Chapter Two

Web and Hyperlinks

2.1 Introduction

This chapter starts by talking about the World Wide Web and its history**,** Web site**,** HTML language and URL which explain how the Web works.

After describing hyperlinks then talking about types of hyperlinks and Anchor tag which is used for defining both source and destination of a hyperlink.

The last part of this chapter describes the problem of broken links and the suggestion to solve it.

2.2 The History of World Wide Web [Jef01]

Fast forward to 1989. A researcher named Tim Berners-Lee, working at the European Particle Physics Laboratory, made a proposal for a simple hypertext system. Hoping to connect the distributed work of physics researchers, Berners-Lee developed a prototype system for linking information including three critical pieces: a way of giving everything a uniform address, a protocol for transmitting these linked bits of information, and finally a language for encoding the information. Working with fellow researcher Mike Sendall, Berners-Lee created both a server for storing and distributing information, as well as a client application for browsing. They called this system "World Wide Web" it had to embody the following characteristics:

1. ***Simplicity:*** Keenly aware of the incredible complexity inherent in Generalized Markup Language (SGML), Berners-Lee opted for a tiny subset of tags for describing a document, and didn't bother with a method for describing a document's styles.
2. ***Universality:*** He imagined dozens, or even hundreds, of hypertext formats in the future, and smart clients that could easily negotiate and translate documents from servers across the Net. While this vision may not have become reality, the fact remains today that HTML and its derivatives can be read on virtually any computer, and on many devices like phones and hand-held units.
3. ***Degradability***: While maintaining a simple system, as well as one that worked across the diversity of the Internet, Berners-Lee realized that HTML would eventually have to expand. To accommodate managed growth, he added a final axiom regarding new versions: they must never break older releases of the language. So as the nascent Web evolved, it would never *require* upgrades. New versions would simply be embellishments of old versions**.**

2.3 Web Page Document

A Web page is a resource of information that is suitable for the World Wide Web and can be accessed through a web browser. This information is usually in HTML or Extensible Hypertext Markup Language XHTML format, and may provide navigation to other web pages via hypertext links.

Web pages may be retrieved from a local computer or from a remote web server. The web server may restrict access only to a private network, e.g. a

corporate intranet, or it may publish pages on the World Wide Web. Web pages are requested and served from web servers using Hypertext Transfer Protocol (HTTP).

Web pages may consist of files of static text stored within the web server's file system (static web pages), or the web server may construct the XHTML for each web page when it is requested by a browser (dynamic web pages).

A Web page is generally a single HTML document, which might include text, graphics, sound files, and hypertext links. Each HTML document created is a single web page, regardless of the length of the document or the amount of information included [Ann99].

The first document or web page that users see when they enter the site is called the home page. The site might also contain additional documents, files or web pages, which are sometimes called **child pages** [Vir01].

The homepage also called index or default page, is the URL or local file that automatically loads when a web browser start to search in the Web site [Sun01].

2.4 Web Site

Web site is a collection of web pages under the control of a particular person or group. Generally, a web site offers a certain amount of organization of its internal information. The user might start with an ***index or default*** **page** for a web site and then use hypertext links to access more detailed information. Another page within the web site might offer links to

other interesting sites on the web, information about the organization, or just about anything else [Jim03].

Any web site is represented by a directory structure. Directories (“places” to store files) are organized into a hierarchical structure that fans out like an upside-down tree. The top-most directory is known as the *root* and is written as a forward slash (/). The root can contain several directories, each of which can contain subdirectories; each of these can contain more subdirectories, and so on. A subdirectory is said to be the "child" of the directory that holds its "parent". Figure (2.1) shows a system with five directories under the root.

The directory *users* have two subdirectories, *jen* and *richard*. Within *jen* are two more subdirectories, *work* and *pers*, and within *pers* is the file art.html [Jen99].

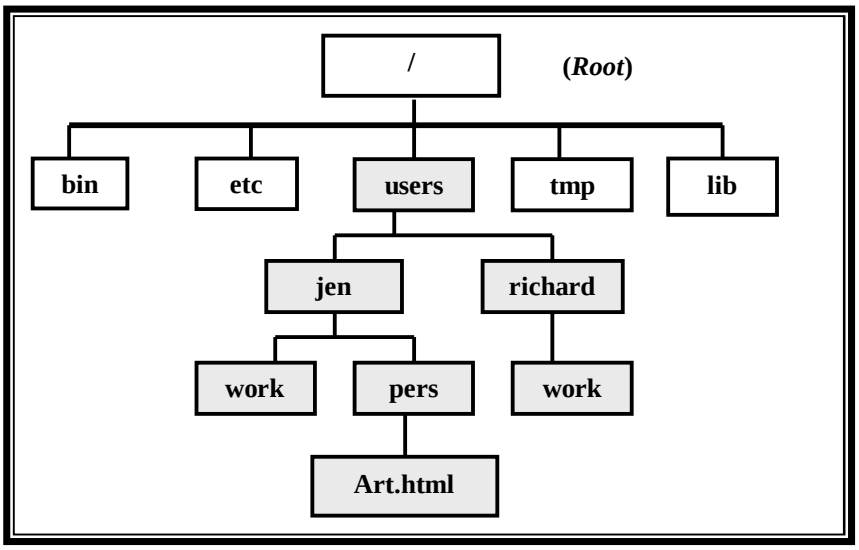


Figure (2.1) The strucutre of web site

2.5 The Uniform Resource Locator

Every document on the World Wide Web has a unique address, this document's address is known as its ***uniform resource locator*** **(URL).**

"URL" usually is pronounced "you are ell". A URL consists of the document's name preceded by the hierarchy of directory names in which the file is stored **(*pathname*)**, the Internet domain name of the server that hosts the file and the software and manner by which the browser and the document's host server communicate to exchange the document (protocol):
protocol://server_domain_name/pathname

Several HTML tags include a URL attribute value, including hyperlinks, inline images, and forms. All use the same URL syntax to specify the location of a web resource, regardless of the type or content of that resource. That's why it's known as a ***uniform resource locator***.

Since it can be used to represent almost any resource on the Internet, URLs come in a variety of flavors. All URLs, however, have the same top-level syntax: scheme: scheme_specific_part.

The scheme describes the kind of object the URL references; the scheme_specific_part is, well, the part that is peculiar to the specific scheme.

The important thing to note is that the *scheme* is always separated from the scheme_specific_part by a colon with no intervening spaces [Chu98].

2.5.1 The Different Protocols for URLs[Ann99]

There are some protocol uses for URL but already mentioned that HTTP is the protocol most often used by Web browsers to access HTML pages. Table (2.1) shows some of the other protocols that can be part of an URL.

Protocol	*Accesses…*
http://	HTML documents
https://	Some "secure" HTML documents
file://	HTML documents on hard drive
ftp://	FTP sites and files
gopher://	Gopher menus and documents
news://	Use Net newsgroups on a particular news server
news:	Use Net newsgroups
mailto:	E-mail messages
telnet:	Remote Telnet (login) session

Table (2.1) Possible Protocols for an URL

By entering one of these protocols, followed by an Internet server address and a path statement, the user can access nearly any document, directory, file, or program available on the Internet or on his (her) hard drive.

2.6 Hypertext Markup Language[Ann99]

Hypertext Markup Language (HTML) developed as a subset of SGML which is a higher-level mark-up language that has long been a favorite of the Department of Defense. Like HTML, it describes formatting and hypertext links, and it defines different components of a document. HTML is definitely the simpler of the two, and although they are related, there are few browsers that support both.

Because HTML was conceived for transmission over the Internet in the form of Web pages, it is much simpler than SGML, which is more of an application-oriented document format. While it's true that many programs

can load, edit, create, and save files in the SGML format (just as many programs can create and save programs in the Microsoft Word format), SGML is not exactly ideal for transmission across the Internet to many different types of computers, users, and browser applications.

HTML is more suited to this task. HTML lets the designer create pages that reasonably sure can be read by the entire population of the Web. Even users who are unable to view page graphics, for instance, can experience the bulk of what it communicating if the designer design HTML pages properly.

At the same time, HTML is a simple enough format that typical computer users can generate HTML documents without the benefit of a special application. Creating a WordPerfect-format document would be rather difficult by hand (including all of the required text size, fonts, page breaks, columns, margins, and other information), even if it weren't a "proprietary"-that is, nonpublic-document format.

HTML is a public standard, and simple enough. This simplicity is part of a trade-off, as HTML-format documents don't offer nearly the precision of control or depth of formatting options that a WordPerfect or Adobe PageMaker-formatted document would.

The specification of links in HTML: [W3c04]

1. HTML links are **not stable**: if the destination document is re (moved) the link points to null.
2. HTML links are **unidirectional**: destination document's do not "know" that a link points at them.

3. HTML links are always **one to one relationships**: there is always one source and one destination. It is not possible to model more complex relationships between documents with a HTML link.
4. HTML links do not support **version management**: as they are hard-coded it is not possible to update the destination to a new version automatically. Further it is not possible to create HTML links that point to a whole bunch of versions of the same document.
5. HTML links do not provide **rights management**: it is not possible to control access to a link. An HTML links is always shown and can be followed by every single user.
6. HTML links do not have **types**: in other words: there is only one type of HTML link. It is not possible to map different semantics of links to different link type.
7. HTML links cannot be annotated by arbitrary **metadata**: a part from the title of a link it is not possible to attach metadata. This makes it impossible to have proper descriptions for semantically complex link relationship.

2.7 Hyperlinks Concept

Unlike any other Internet service or protocol, the World Wide Web is based on a concept of information retrieval called **hypertext.** In a hypertext document, Hyperlinks are links in a web page that could be clicked on to go to other web resources. The user can also do thing like open an e-mail program to send messages for anyone. When the cursor is on hyperlinks, it changes into the shape of a hand. Usually, a text hyperlink is of a different color than regular text and underlined. Images can also be hyperlinks,

hyperlinks are also called links. Without links, websites would be virtually worthless and very boring [Ann99].

The true power of HTML, however, lies in its ability to join collections of documents together into a full library of information, and to link documents with other collections around the world. Just as readers have considerable control over how the document looks onscreen, with hyperlinks they also have control over the order of presentation as they navigate through their information[Chu98].

Hypertext gives users the ability to retrieve and display a different document simply by a click of the keyboard or mouse on an associated word or phrase hyperlink in HTML document.

Use these interactive hyperlinks to help readers easily navigate and find information of otherwise separate documents in a variety of formats, including multimedia, HTML, and plain ASCII text.

To include a hyperlink to some other document in web collection, all the user need to know is the document's unique address and how to drop an *anchor* into he (she) HTML document [Jen99].

Hypertext is non-sequential by definition. There is no single order that determines the sequence in which the text is to be read. It provides a means for readers to actively explore rather than passively absorb a body of information. Hypertext consists of interlinked pieces of text (or other types of information). Each unit of information is called a node. Whatever the grain size of these nodes, each of them may have pointers to other units, and these pointers are called links.

Figure (2.2) shows that the entire hypertext structure forms a network of nodes and links. Readers move through this network in an activity called

browsing or navigating, rather than just "reading", to emphasize that users must actively determine the order in which they read the nodes.

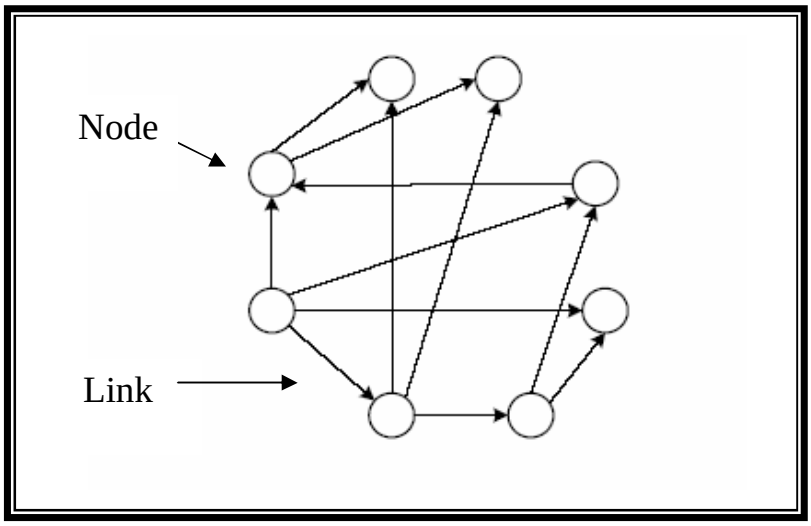


Figure (2.2) Example of hypertext structure

User interfaces of hypertext systems give users (limited) freedom to navigate through the hyperspace. (Hyperspace is a term used for the structure formed by the nodes and links). Users only need to e. g. "click" on the links to go anywhere they want, and in the meantime users get a hyperdocument instead of a plain text document. Hypertext systems provide users a little or a large amount of navigational freedom; depending on how "rich" the link structure is [Hon02].

2.8 Creating Link by Using Anchor Tag [Ann99]

The anchor (<A>) tag is the HTML feature for defining both the source and the destination of a hyperlink. The user will most often see and use the <A> tag with its HyperRefrence attributes (HREF) to define a source hyperlink. The value of the attribute is the URL of the destination. The contents of the source <A> tag is the words or images between it and its

</A> end tag is the portion of the HTML document that is specially activated in the browser display and that users select to take a hyperlink. These *anchor* contents usually look different from the surrounding content (text in a different color or underlined, images with specially colored borders, or other effects), and the mouse pointer icon changes when passed over them. The <A> tag contents, therefore, should be text or an image (icons are great) that explicitly or intuitively tells users where the hyperlink will take them. For instance, the browser will specially display and change the mouse pointer when it passes over the text, for examples:

To a local file:

<A HREF="filename.html">...</A>

To an external file:

<A HREF="http://server/path/file.html">...</A>

To a named anchor:

<A HREF="http://server/path/file.html#fragment">...</A>

To a named anchor in the same file:

<A HREF="#fragment">...</A>

To send an email message:

<A HREF="mailto:username@domain">...</A>

To a file on an FTP server:

<A HREF="ftp://server/path/filename">...</A>

2.9 Types of Hyperlinks

There are three types of HTML hyperlinks, and each one is used in different situation.

1. **Absolute URLs**: links to a page on a different web server.

2. **Relative URLs**: links to a page on the same web server.
3. **Linking within a document**: links to a different location on same web page.
    a. Linking to fragment.
    b. Linking to fragment in another document.

Figure (2.3) illustrates the difference between relative and absolute URLs.

If the user creates a web site of any complexity, he (she) will need all three types of hyperlinks.

All hyperlinks have two components:

1. A **link label** (a clickable element on a web page).
2. A **link destination** (a target destination).

The link label and the link destination are both components found inside the special HTML element used to create hyperlinks. Here's example preview of a link in HTML:

<A HREF="http://www-edlab.cs.umass.edu/cs120/">cs120 homepage</A>

Link destination **Link label**

Armed with these two components, the users can add any links needed to their web pages. A *link label* can consist of any visible element on a web page.

Although it is usually text (clickable text), it can also be an image (a clickable image). A *link destination* is usually another .html file, but it can also be any file (not necessarily an html file) that the web page author has chosen to distribute over the web [Dav00].

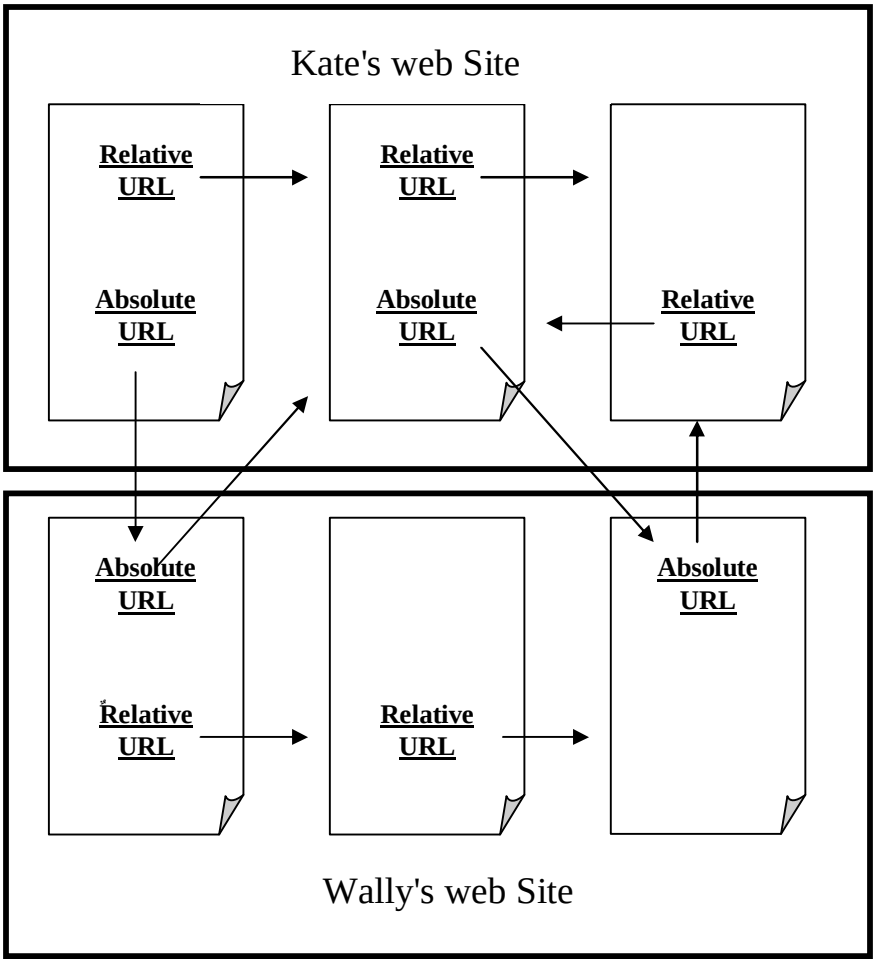


Figure (2.3) Relative and absolute URLs

2.9.1 Absolute URLs[Vir01]

An absolute URL is made up of the following components: a protocol identifier, a host name (the name of the server machine), the pathname (if there is one), and the specific file name. When users are linking to documents on other servers, they need to use an absolute URL. The following is an example of a link with an absolute URL:

<A HREF="http://www.littlechair.com/web/index.html">...</A>

Here the protocol is identified as http (the standard protocol of the Web), the host is www.littlechair.com and web/index.html is the pathname leading to the particular file.

2.9.2 Relative URLs [Vir01]

A relative URL provides a pointer to another document relative to the location of the current document. The syntax is based on relative pathname structures in the UNIX operating system.

When users are pointing to another document within their own site (on the same server), it is usually best to use relative URLs.

For example, if user currently in jcc.html (identified here by its absolute pathname): www.littlechair.com/web/samples/jcc.html.

If user wants to put a link on that page to talk.html, which is in the same directory: www.littlechair.com/web/samples/talk.html.

Using a relative URL within the link as follows:

```
<A HREF="talk.html">...</A>
```

Using the same example, to link to the file index.html in a higher level directory (web), use the relative pathname to that file as shown:

```
<A HREF="index.html">
```

This relative URL is the equivalent to the absolute URL

http://www.littlechair.com/web/index.html.

2.9.3. Linking Within a Document

By default, when user links to a page, the browser displays the top of that page. To aid in navigation, the user can use the anchor tag to link to a specific point or section within a same document or within another document in same web site.

1. Naming a Fragment within Same document

a. Naming a Fragment

First, the user need to identify and name the portion of the document (called a fragment) that wanted to link to. The fragment is marked using the anchor (<A>) tag with its name attribute, giving the document fragment a name that can be referenced from a link.

To illustrate, let's set up a named fragment within a sample document called Dailynews.html so users can link directly to the Stock Quotes section of the page.

The following anchor tag marks the Stock Quotes title as a fragment named "stocks."

```
<A NAME="stocks">Daily Stock Quotes</A>
```

b. Linking to a Fragment

The second step is to create a link to the fragment using a standard anchor tag with its HREF attribute. Fragment identifiers are placed at the end of the pathname and are preceded by the hash (#) symbol.

To link to the "stocks" fragment from within dailynews.html, the link would look like this:

```
<A HREF="#stocks">Check out the Stock Quotes</A>
```

2. Linking to a Fragment in another Document

The user can create a link to a named fragment of any document on the Web by using the complete pathname. (Of course, the named anchors would have to be in place already.) To link to the stocks section from another document in the same directory, use a relative pathname as follows:

```
<A HREF="dailynews.html#stocks">Go to today's Stock Quotes</A>
```

[Jen99]

2.10 Simple and Graphical Hyperlinks Structure

Hyperlinks may represented by text, graphics, list or other structure types, in this research using text and graphics structure:

1. Simple Hypertext Links

The anchor (<a>) tag is used to identify a string of text that serves as a hypertext link to another document. In its simplest incarnation, it looks like this:

<A HREF="news.html">The News </A>

2. Graphical Links

Graphics work as well as all other types of HTML tags. Simply by placing an <IMG> tag inside an anchor tag; the user could be creating a clickable image, which can substitute for the descriptive text in a link. For example:

<A HREF="http://www.fakecorp.com/"> <IMG SRC="biglogo.gif" ALT="Bigcorp"></A>

Notice that the example doesn't include any sort of descriptive text in the link. If a user's graphical viewer can support this type of image, the link displays the graphic, with a colored border. Clicking the image sends the browser to the associated link. If the user isn't viewing this page with a graphical viewer, he or she sees the ALT text, which works as a hyperlink. [Ann99].

2.11 404 Error Message[Ger99]

404 is an HTTP status code, anyone who uses a computer on a regular basis has been faced with a 404 error message.

The 404 error message commonly appears when users request a URL that the server does not have. For some reason, the server chooses to simply flash a 404 error message rather than providing them with the information that they need. All they know from the information provided is that the page has not been found. This is usually a default message that most Web servers return when someone requests a URL that the server does not have.

Every time the user visits a web page, his (her) computer (client) is requesting data from a server using HTTP, or Hypertext Transfer Protocol. Before the web page is even displayed in his (her) browser, the web server has sent the HTTP header, which contains the status code.

For a normal web page, the status is **200 OK**. The user doesn't see this because the server proceeds to send his (her) the contents of the page. It's only when user encounters an error that sees the actual status code, such as 404 not found. HTTP status codes were established by the World Wide Web Consortium (W3C) in 1992. They were defined by Tim Berners-Lee, the same person who single-handedly invented the web and the first web browser in 1990.

Code 404 is meaning that, the first 4 code indicates a client error. The server is saying that user have done something wrong, such as misspell the URL or request a page which is no longer there. The middle 0 code refers to a general syntax error. This could indicate a spelling mistake.The last 4 code just indicates the specific error in the group of 40x, which also includes 400: Bad Request, 401: Unauthorized, etc.

2.12 The Broken Link's Problem

In hypertext systems such as the World Wide Web, links are a vital component to the navigation between and within Web pages on the Internet.

A common problem in Web pages is the presence of broken links, broken links occurs when a link points to a file that can not be accessed.

Based upon a literature of surveys performed on variety Web professionals, the most common reasons broken links occur are the moving or renaming of linked files or the misspelling of the path name in the link itself, hardly a single Web professional has not experienced some sort of difficulty caused by sources external to their area of responsibility and outside of their control. Difficulties such as shared network resources are being changed, including folders being renamed or network drive mappings being altered. Reorganization of directory structures alone would wreak havoc on a Web site and the links contained within it. Every Web professional involved with the development and management of a Web site has to address the problem of broken links. As the Cyber economy grows, the task of keeping links accurate and up-to-date becomes even more crucial.

According to Nielsen ratings, as of May 2002 there were 580.78 million people online worldwide. In 2001, Google — the leading Internet search engine — had access to over 1.5 million URLs which contain over 2.4 billion Web pages. If we conservatively estimate that each Web page on the Internet has at least one broken link, then in 2.4 billion instances, information intended to be exchanged on a Web site would not be available, and in many of these cases visitors would be prevented from successfully navigating the site.

A Web site, if not properly maintained, can become a liability, even when a site is regularly maintained, problems still can occur.

Broken links, when not found have caused frustrated and irritated users, lost customers and lost revenue. As a result, Web developers and Webmasters currently spend a significant amount of their valuable production time manually maintaining links within their Web sites in an effort to reduce the number of lost visitors and, therefore, lost profits [Lin02].

A **dead link** or **broken link** is a link on the World Wide Web that points to a web page or server that is permanently unavailable. The most common result of a dead link is a 404 error, which indicates that the web server responded, but the specific page could not be found. The browser may also return a Domain Name Server (DNS) error indicating that a web server could not be found at that domain name.

Another type of dead link is a URL that points to a site unrelated to the content sought. This can sometimes occur when a domain name is allowed to lapse, and is subsequently reregistered by another party. Domain names acquired in this manner are attractive to those who wish to take advantage of the stream of unsuspecting surfers that will inflate hit counters and Page Ranking [Gre99].

Twelve percent of the sites' links did not lead to the information promised, a condition that damages the logos appeal of the site. While most designers test for broken links, the content analysis located them more comprehensively [Hea05].

2.12.1 Disadvantages of Broken Link's Problem

One of the most important indicators of a high-quality site is absence of broken links.

1-There is nothing worse for a user than coming across either a link that leads nowhere or an empty square instead of an image or video.

2-Broken links ruin web site reputation and bring down its rating on search engines.

Broken links are not just errors in the design of a site but they can cost rating, traffic and money **[**Jef01].

The interconnected, hypertext character of the Web contributes significantly to its importance. It allows for rapid changes to be made to files, easy connections between one site and another and detailed intrasite navigational features. The Web encourages rapid, frequent, and direct publication of information. At the same time, it means that just as rapidly, pages can disappear, move, and change their entire content.

The links on Web pages connect to other Web pages. Some of the pages could be on the same Web server while others can be anywhere else on the Web. This great strength of the Web is also a weakness, in that a company can never know every Internet site, intranet page, or bookmark list that links to its pages. And thus, it is impossible to update every page linking to its pages.

A Web page dies every time that one or more files on a Web server have their names changed, their location in the subdirectory structure moved, or their host names modified. If a file named FIRSTtry.htm becomes redesign.html, anyone linking to the old URL will get the "404 File not

Found" message. When "www.name.org/directory/subdirectory/file.html" becomes "www.name.org/directory/file.html"; the old URL is again left orphaned.

Links also die when a server name changes. When "www.host.net/company" becomes "www.company.com" and no redirect files are left behind, all the links to the old host die. Then again, there are plenty of pages, directories, and sites that have just been removed. An organization may cease to exist, stop paying its Web bills, or get cut off by its hosting company. Once again, all the links pointing to those pages then die [Gre99].

Diomiidis Spinellis group studied over four thousand references to WWW resources collected from research papers published in communications of the Associated Colleges of Midwest (ACM) and Institute of Electrical and Electronics Engineers Standard Association (IEEE) computer society journals from 1995 through 1999 and results was as below the biggest source of unavailable references was 404 Not Found errors, according for 21.82% of the total. DNS resolution was the second biggest problem, which caused 11.32% of the references to not fail.

From the Figure (2.4) shown below is clear to see that there is a strong correlation to year and the number of DNS errors, while there is only weak correlation to 404 Not Found errors which stayed mostly the same from year to year [Rya00].

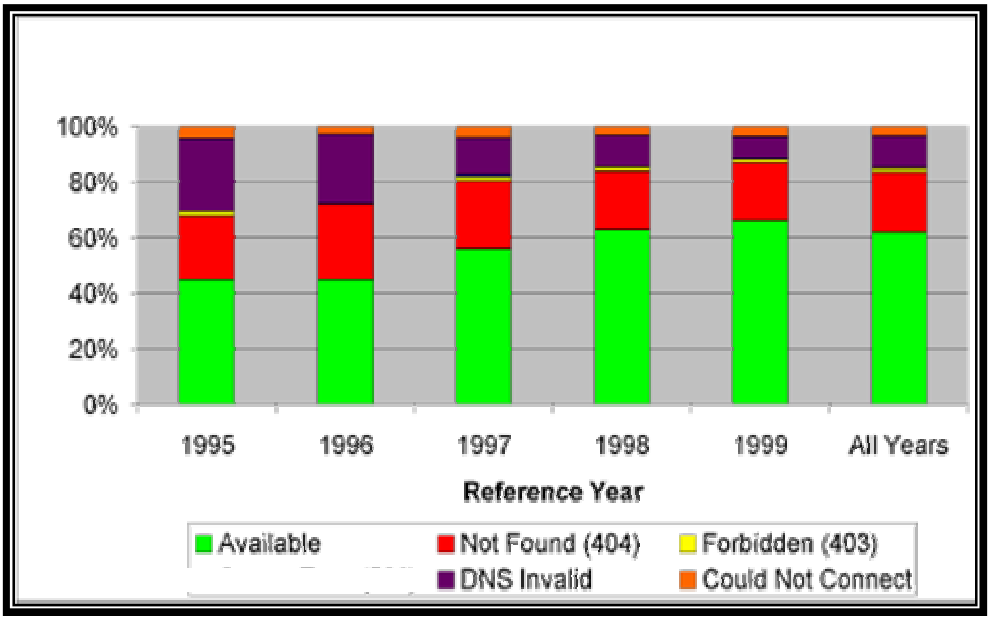


Figure (2.4) Availability of Reference in WWW

Broken links have hitherto been a source of constant annoyance for Web professionals who have been charged with maintaining the integrity of links within Web sites. Surveys conducted by **LinkTek Corporation** have served to uncover the scope of the problem of broken links.

When Web professionals were asked what the resulting problems caused by broken links there are found as follows:

1. 71% reported that visitors were unable to find what they were looking for.
2. 69% reported that visitors were unable to properly navigate the site.
3. 57% reported that visitors became frustrated and irritated due to broken links.

When asked what methods Web professionals currently use to fix broken links in a Web site:

1. 88% reported that they manually correct broken links on a regular basis.

2. The remaining 12% reported using a Web authoring program to assist with the finding and manual fixing of broken links.

And when they asked what actions Web professionals take to prevent broken links from occurring:

- 74% reported that they perform manual link checking and verification on a regular basis.

Given the ubiquity of the problem, along with the existing inefficient manual methods of maintenance and repair, the need for a productivity enhancing application that could not only find but also repair broken links became obvious [Lin02].

2.13 The Basic Crawling Algorithm

Web crawling is a process to collect all the web pages that are interests [Jia00].

Crawlers are also called robots, spiders, worms, wander walkers, and know bots. The first crawler, Wanderer was developed by Matthew Gray in 1993. Due to the competitive nature of the search engine business; the designs of these crawlers have not been publicly described. There are several crawling techniques available in public. The simplest one is to start with a set of URLs and from that extracts other URLs recursively in breadth-first or depth-first manner [Sun01].

The crawler consists of a URL list, downloading program and a URL extractor. Before running the crawler designers will have to initialize the URL list with some URLs (Home page and other frequently requested URLs). The downloading program contains a pointer that points to one of the links in the URL list, which will be initialized to zero.

When the crawler start running, the downloading program will increase the pointer by one and check if there is a link at the pointed position in the URL list. If there is a link, then the downloading program will request to download that link from the internet and save it as a file on the storage unit. If there was no link then this means that there are no new links in the URL list, which is the end of the crawling process. After downloading a page from the internet, it will be send to the URL extractor, which will extract all the links inside it and these links will be added to the URL list, but without duplicates (only the new links are added). The crawler complete it's work when there will be no more new links are added to the URL list and the pointer of the downloading program cross the last location in the URL list. Figure (2.5) shows the whole process.

Running a web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues [SL98].

The design of a good crawler presents many challenges. Externally, the crawler must avoid overloading Web sites or network links as it goes about its business. Internally, the crawler must deal with huge volumes of data.

Unless it has unlimited computing resources and unlimited time, it must carefully decide what URLs to scan and in what order [JHL98].

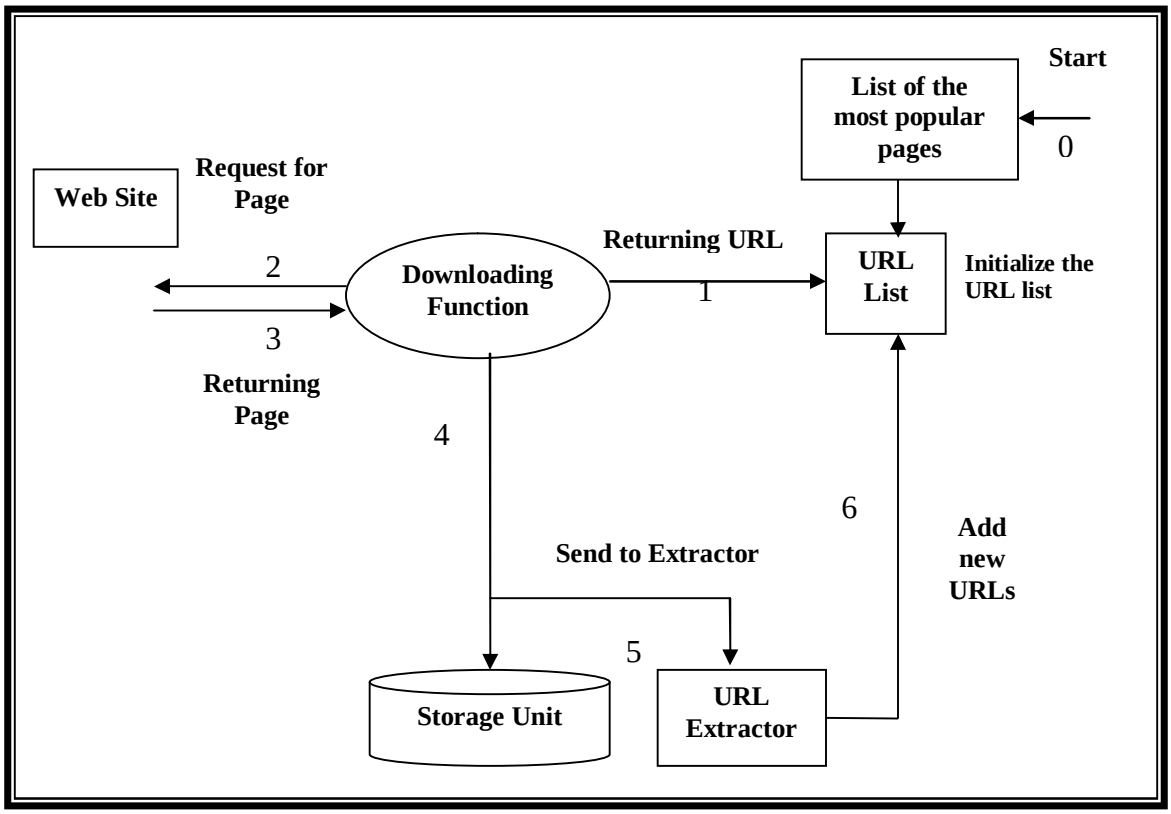


Figure (2.5) Crawling Process

2.13.1 Crawling Policies

There are two policies used to traverse web pages. The first one is **breadth-first policy**. It looks at all the pages linked by the current page and so on. The coverage will be wide but shallow. This may cause the web server to have many rapid requests. The second is **depth-first policy**. The users follow the first link of a page and they do the same on that page until they cannot go deeper. After that, it returns recursively. The advantage of using depth-first search is deep and space complexity is cheaper. But disadvantage of using it is narrow. Consider the web site shown in Figure (2.5).

If the crawler is using a breadth-first policy then the pages will be downloaded in the order (home, page1, page2, page3, page4, page5, page6, page7, page8, page9, page10, page11). If the crawler is using a depth-first policy then the pages will be downloaded in the order (Home, page1, page4, page5, page10, page11, page6, page2, page7, page8, page3, page9)[Eih05].

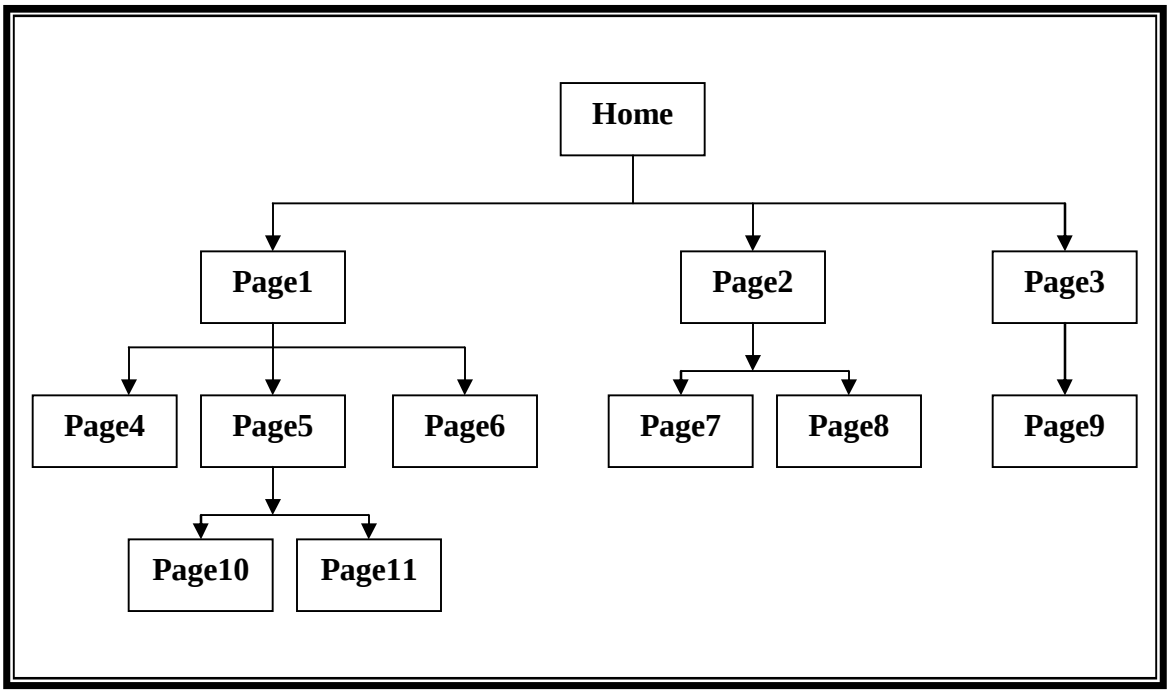


Figure (2.5) website example

Chapter Three

Design of proposed Link Checker Software

Chapter Three

Design of Proposed Link Checker Software

3.1 Introduction

This chapter is organized as follows: at first, in section (3.2), software architecture is explained. Section (3.3) illustrates the crawler module. In section (3.4), image link extractor module explained in detail. Section (3.5), illustrates the checker module.

3.2 Proposed Software Architecture

This thesis aim to design and implement software that test website links, and gives a report about broken links, whether these links were text links or image links. The structure of the proposed link checker software consists of three main modules as shown in Figure (3.1), each module has specific functions.

These three modules are:

1. **Crawler and extractor module**: this module accesses all the pages of website starting with home page, and then extracts all the text links founded within web pages, then they classified them according to their types (absolute link, relative link, fragment link).
2. **Image link extractor module:** all image links in web pages are extracted in this module**.**
3. **Checker module:** this module consists of two sub-modules:
    a. **Text links checker module**: all fragment links and absolute links are checked.

b. Image links checker module: all extracted image links are checked.

The above mentioned modules will be discussed in details in the next sections. Figure (3.1) show the general structure of proposed link checker software.

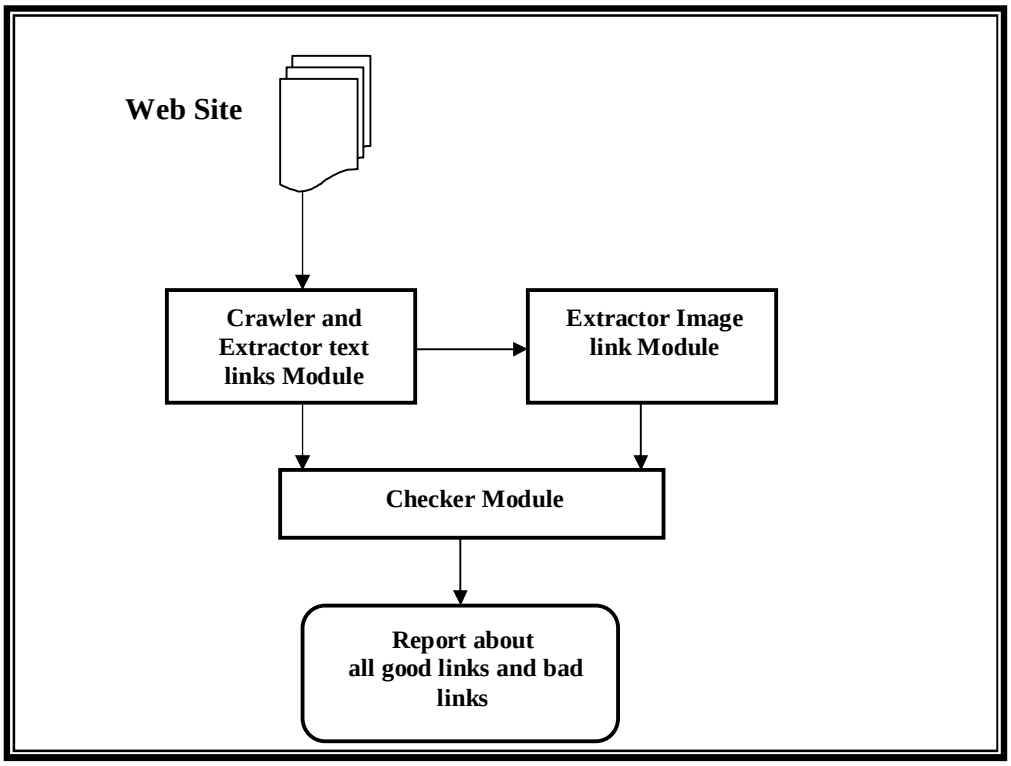


Figure (3.1) Proposed link checker software modules

3.3 Crawler and Extractor Module

The crawling is a process to collect all web pages. There are several crawling techniques available in public. The simplest one is to start with a set of URLs and from that extract other URLs recursively in breadth-first or depth-first manner, in this proposed software a depth first search is used because it is less than in complexity.

The crawler consists of two parts in this proposed link checker; as shown in Figure (3.2).

These two parts are:

1. **Initializing the crawling information**: All information related to the crawling process is initialized, this information represents by three lists of records (relative list, absolute list and fragment list), size of these lists and the pointer which is used to locate current link.

2. **Extracting links and classified them to their types**: In this step the crawling fetch links one by one from relative list then open it as file for reading to extracting all text links and classified to their three types (relative list, absolute list and fragment list).

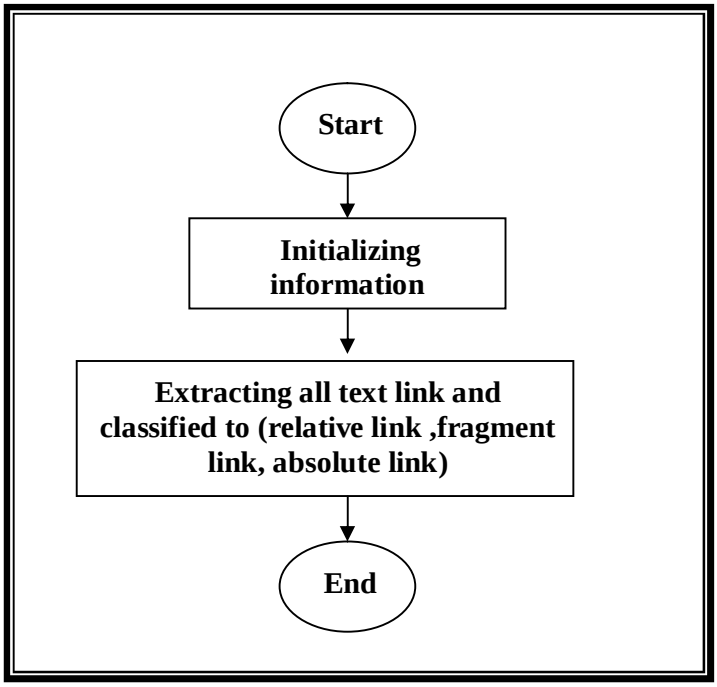


Figure (3.2) Crawler module

3.3.1 Initializing the Crawler Information

The crawler considers that relative list is the main list which contains the first page in website (home page).

The information of the crawling process contains the following variables, which will be initialized to *zero* as shown in Algorithm (3.1).

1. **Size-relative, size-absolute and size-fragment**: Each one of these variables initialized to *zero*.
2. **Relative-list, absolute-list and fragment-list:** Each one of these lists is initialized to empty.
3. **Current-link**: Initialized the index of current link to *zero.*

Relative-list, fragment-list and absolute-list are lists of records each record consists of five fields:

1. **Link field**: Contains the name of the link.
2. **Found field**: Contain boolean value (true or false) which explain if the link found in the website or not.
3. **Check field**: The crawler prevent duplication in link lists (only new links are added) so this field needed, contain boolean value (true or false) which explain if current link checked or not (i.e. if already exist in relative-list).
4. **Source-page field**: Contains the name of page which the link extracted from it.
5. **Destination-page field**: Contains the name of destination page which the link pointed to.

Algorithm (3.1) startup Algorithm

Goal: Initializing crawler information.
Input: home page of a website.
Output: relative-list with the first link value.

Set relative-list (current-link) of link field ← home page
Set relative-list (current-link) of check field ← false
Set relative-list (current-link) of page-destination field ← name of home page

After initializing relative-list with the name of home page in previous Algorithm, now fetching links from relative-list one by one begins with home page, after each page is checked, all links in that page are extracted out and then classified them to their types (i.e. if the link is relative link put it in relative-list, if the link is fragment link put it in fragment-list, if the link is absolute link put it in absolute-list).

The crawler finishes its work when there is no more links in relative link list and the current link point to the last link (see Algorithm (3.2)).

Algorithm (3.2) is the main Algorithm for crawling web pages.

Algorithm (3.2) crawling web pages

Goal: Traverse relative-list starting with home page
Input: all pages of web site
Output: relative-list, absolute-list and fragment list
Variables: link, fullpath as string

```
While (current-link ≤ size-relative) do
  Set link ← relative-list (current-link).link
  Set fullpath-link execute concat-root(link)
  Execute search-page(link)          //check If link found or not
    If search-page(link) = false then
        Set relative-list (current-link).found ← false
    Else
        Set relative-list (current-link).found← true
    End If
    If relative-list (current-link).check equal false then
       Execute read-page(link)       //to extract all links  and classified
       Set relative-list(current-link).check← true
    End If
End While
```

The crawling Algorithm calls three Algorithms:

1. **Concat-root Algorithm**.
2. **Search-page Algorithm**.
3. **Read-page Algorithm**.

3.3.2 Concat-root Algorithm

Link's syntax in HTML language is written without writing root directory, so this Algorithm is used to get fullpath of link, then by this fullpath of link the crawling Algorithm could reach exact links.

For example:

Let's website named al-nahrin university which is represent the same name of root directory exist in drive D:\.

Let's relative link "Msc-student.html" exists in dept-computer-science subdirectory, so the full path of that page is "**D:\al-nahrin university\ dept-computer-science\Msc-student.html**".

Algorithm (3.3) concat-root

Goal: Get fullpath of link
Input: link
Output: fullpath-link
Variables: root as string

```
Set root ← "drive-name:\directory"
  If  first character of link = "\"  then
      Set fullpath-link ←root + link   //means the page in root directory
  Else
     Set fullpath-link ← root+ "\" + link
  End If
Return fullpath-link
```

3.3.3 Search-page Algorithm

This Algorithm used to check if page is actually exists within website or not.

Algorithm (3.4) Search-page

Goal: Check If page exist within Web site or not
Input: page-name
Output: True or False
Variable: path

Set path←Dir(page-name)
If path ≠ " " ***then***
Return true
Else
Return false
End if

3.3.4 Read-page Algorithm

This Algorithm considered as main and important Algorithm of this proposed software, is used to extract all text links and classified them to their types. See Algorithm (3.5).

After each link extraction, read-page Algorithm calls two Algorithms to classify extracted link to its type. These are:

a. **Is-Fragment Algorithm**: this Algorithm is used to check if link is fragment or not. See Algorithm(3.9)
b. **Is-Absolute Algorithm**: this Algorithm is used to check if link is absolute or not. See Algorithm(3.10)

Finally, if link is neither fragment link or absolute link this means it is relative link. To avoid duplicate same link in one list, so that Check-exist Algorithm was used. See Algorithm (3.8)

Algorithm (3.5) Read-page

Goal: Open page as file to extracted all links and classified them to there types.
Input: page-name
Output: Relative-list, Absolute-list and Fragment-list
Variables: ch as character; HR, st, link, fragment, destination-page as string; absolute-flag, flag-exist as boolean

```
Open page as file
  While not end of file
      Read character(ch) from file
      If ch = "<" then
          Read ch from file
          If ch = "A" OR ch = "a" then      //found <A\> tag
              Read ch from file
              If ch = " " then
                  Set HR ← " "                  //found HREF attribute
                  Read ch from file
                  While ch ≠ "="
                      Set HR ← HR + ch
                      Read ch from file
                  End while
              End If

          If execute (check-HREF) = true then
              Set st ← " "
              While ch ≠ " " " "
                  Read ch from file
                  Set st ← st+ch
              End While
          End If
        While ch ≠ " " " character
          Read ch from file
          If ch = " "" character then
              Set st ← st+ch
          End If
        End While
  Set link ← st
  Set destination-page← " "
```

To be continue

```
Set fragment ← execute Is-fragment (page-name, link, destination-page)
  If  fragment ≠ " " then
     Set fragment-list(size-fragment).link←  fragment
     Set fragment-list(size-fragment).found← false
     Set fragment-list(size-fragment).check← false
     Set fragment-list(size-fragment).source-page← page name
     Set fragment-list(size-fragment).destination-page← page-destination
     Set size-fragment ← size-fragment+1
 End If

 Else
 Set absolute-flag ← execute is-absolute (link)
  If absolute-flag = true then
    If execute check-exist(absolute-list) = false then
         Set absolute-list (size-absolute).link← link
         Set absolute-list (size-absolute).source-page← page-name
         Set absolute-list (size-absolute).destination-page← link
         Set size-absolute ← size-absolute+1
     Else
         Exit If
    End If
  End If
 Else
    Set flag-exist ← execute check-exist(link)
     If  flag-exist =l false then
         Set size-relative ← size-relative+1
         Set relative-list(size-relative).link← link
          Set relative-list(size-relative).source-page← page-name
         Set relative-list(size-relative).destination-page← link
    End If
  End If
End If
End while
Close file
```

Algorithm (3.7) illustrates the process of checking "HREF" attribute which means hypertext references.

Algorithm (3.7) Check-HREF

Goal: Check HREF attribute
Input: HR
Output: True or False

Convert all characters in HR string to small characters
If HR = "href" ***then***
  Return true
Else
  Return false
End If

Algorithm (3.8) Check-exist
Goal: Check if page is found in list or not
Input: page-name, list, size //list is either (relative list or absolute list)
//size is either relative size or absolute size
Output: True or False
Variables: i as integer; flag as boolean.

Set flag ← false
For i = 0 to size-1
  If list(i).link = page-name ***then***
    Set flag ← true
    Exit For loop
  End If
End For
Return flag

3.3.4.1 Is-Fragment Algorithm

This Algorithm depends on "#" character to check if this link is fragment link or not. See Algorithm (3.9)

Algorithm (3.9) Is-Fragment

Goal: Check link If fragment link or not
***Input: link, page-name,* destination-page**
Output: fragment-name
***Variables*: fragment-link as string; i, j as integer; ch as character**

```
Set fragment-link ← " "
Set destination-page ←" "
If first character of link string = "#" then    //fragment link is linking to
                                                  fragment
   For i = 1 to length (link)
      Read ch
      Set fragment-link ←fragment-link+ch
   End For
     Set destination-page ←page-name
Else
    For i = 1 to length(link)
     Read ch from link
     If ch ="#" then  //fragment linking to fragment in another documen
        For j = i+1 to length(link)
            Read ch
            Set fragment-link ← fragment-link+ch
        End For
        For i = 0 to position (#)
            Read ch
            Set destination-page ←destination-page+ch
        End For
    End If
End If
 Return fragment-link
 Return destination-page
```

3.3.4.2 Is-Absolute Algorithm

The Checking absolute link represented by check syntax of it's. The URL of absolute link (External website) begins with one of these five protocols (http, news, mailto, gopher, ftp, www). Algorithm (3.10) show the link checking if it's absolute or not.

Algorithm (3.10) Is-absolute

Goal: Check link if it is absolute or not
Input: link
Output: True or False

```
If  (first 7 characters of link equal "http:\\")    OR
    (first 7 characters of link equal "news:\\")  OR
    (first 4 characters of link equal "www.")    OR
    (first 9 characters of link equal "mailto:\\") OR
    (first 9 characters of link equal "gopher:\\")OR
    (first 6 character of link equal "ftp:\\") then
     Return true
 Else
     Return false
End If
```

3.4 Image Link Extractor Module

The proposed link checker software traverses and checks the image links also, after traversing all three types of text links(relative-links, absolute links and fragment links).The proposed software extracted an image link from relative link list because this list contains all pages in a Web site.

In Algorithm (3.11) at first fetch all links one by one from relative list until no more links in it's to extract the image.

The information of the checking image link contains the following variables, which initialized to zero.

1. **Size of image-list**: it is the size of image list.
2. **Image-list:** it is a list of records each record consists of four fields, These are:
   a. **Link field**: contains the name of image.
   b. **Found field**: contains boolean value (true or false) which explains if link found or not.

c. Check field: this field contain boolean value (true or false) which explain if current link checked or not (i.e. if already exist in relative-list).

d. Source-page field: contains the name of page which the link extracted from it.

Algorithm (3.11) send relative link to another Algorithm which was extract-image link Algorithm (3.12).

Algorithm (3.11) stopped while no more link in relative link list.

Algorithm (3.11) Fetch-relative link

Goal: Fetch relative link from relative list to found image link.
Input: Relative link list
Output: image-list
Variables: i as integer; link, page-name as string

```
For i = 0 to (size-relative)-1
    If relative-list(i).found = true then
        Set link ← relative-list(i).link
        Set page-name execute concat-root(link)
        Execute extract-image-link(page-name)
    End If
End For
```

In Algorithm (3.11), number of steps will be done to extract image link.

Algorithm (3.12) Extract-image link

Goal: Traverse image link and check If found or not
Input: page-name
Output: image-list
***Variables*: ch as character; img-tag, scr-tag, img-name as string**

```
Open page-name as file
 While not end of file
   Read character from file
```

To be continue

```
    If ch = "<" then
        Set img-tag ← " "
        For i = 1 to 3              // find IMG tag
            Read character from file
            Set img-tag ← img-tag+ch
        End For
        If  img-tag equal "img" then
           Read ch from file
            If ch = " " then
               Set scr-tag ← " "
               For i =1 to 3              //find SCR tag
                   Read ch from file
                   Set scr-tag ← scr-tag+ch
               End For
            End If
            If  scr-tag = "scr" then
               Read ch from file
               While ch ≠ " " "
                    Read ch from file
               End While
                   Set img-name ← " "
                   Read ch from file
                   Set img-name ← img-name+ch
               While ch not equal (")
                    Read ch from file
                    If ch = " " " then   Exit do
              End While
            End If
       Set img-name ← img-name +ch
     If execute check_image(img-name, source-page) =false Then
         Set image-list(img-size).link ← img-name
         Set image-list(img-size).check ← True
         Set image-list(img-size).found ← True
         Set image-list(img-size).source-page← page-name
         Set img-size ← img-size + 1
     End If
   End If
 End If
End While
Close file
```

To prevent duplicated same image link in image-list the software check if image checked previously or not. See Algorithm (3.13)

Algorithm (3.13) Check-img

Goal: Check if image checked previously or not
Input: img-name, source-page
Output: True or False
Variables: i as integer

```
For i = 0 to img-size - 1
    If (((image-list(i).link = img_name) And
        (image-list(i).source-pages = source-page)) OR
        (image-list(i).link = img_name)) Then
          Return  true
          Exit For
    Else
          Return   False
    End If
End For
```

3.5 Checker Module

After all links (text links or image links) extracted, the proposed software checks to satisfy if these links found or not.

This module consists of two sub-modules:

1. ***Text links checker module***: in this module all fragment were checked.
2. ***Image links checker module***: all extracted image link were checked in this module.

3.5.1 Text Links Checker Module

Since, relative links represents web pages and all links (either text or image) were extracted, so it must be checked if are really exists in Web

site then the reading operation and links extraction operation were done. So checking relative links was done during crawler process.

Associated with, since the software is working offline the absolute links will be checked for syntax only. Therefore the checker module will work on fragments links.

At first the proposed software fetch all fragment links one by one as shown in Algorithm (3.14) to send each one to another Algorithm (3.15) to check if found or not.

Algorithm (3.14) Travers-fragment link

Goal: fetch all fragment links from fragment-list to send to Algorithm (3.15) and update fragment list
Input: fragment-list
Output: fragment-list with update information
Variables: i as integer; destination-page, link as string

```
For i = 0 to size-fragment -1
    Set destination-page ← fragment-list(i).destination-page
    Set link ← fragment-list(i).link
    If execute check-fragment (link,destination-page) = true then
        Set fragment-list(i).found← true //fragment link is found
    End If
End For
```

Algorithm (3.15) using the name of destination-page which sent by Algorithm (3.14) to open it as file for reading to search on fragment link, if found the returned value is true else is false.

Algorithm (3.15)check-fragment

Goal: Check fragment link if found or not
Input: fragment-link, destination-page
Output: True or False
Variables: ch as character; tag, st as string

```
Open destination-page as file
```

To be continue

```
If execute search-page (destination-page) = false  then
      //check if destination-page found in website or not
       Return false
      Exit
End If
While not end of file
    Read ch from file
    If ch = "<" then    // find A tag
        Read ch from file
          If ch = "A" OR ch ="a" then
              Read ch from file
              If ch = " " then
                   Set tag ← " "
                   Read ch from file
                   While ch ≠ "="
                         Set tag ← tag+ch        //find NAME tag
                  End While
                  If (tag = "name") OR (tag = "NAME") then
                        Read character from file
                        Set st ← " "
                        While ch not equal (')
                             Set st ← st +ch
                             Read character from file
                             If st ←link then   //if link s the same word  after
                                                      NAME tag
                                    Return true
                               Else
                                    Return false
                            End If
                        End While
                    End If
               End If
          End If
     End If
End While
Close file
```

3.5.2 Image Links Checker Module

All extracted Image links are check in this module, Algorithm (3.16) imagelink-checking used to checking if image exist in web site or not.

Algorithm (3.16) Imagelink-checking

Goal: check if image link is found in website or not
Input: image-list
Output: updated image-list
Variables: fullpath-img as string; i as integer

```
For i = 0 to img-size -1
  Set Fullpath-img ← concat-root(image-list(i).link)
  If  execute search_img(fullpath-img) = true then
      Set image-list (img-size).found ← True
  Else
      Set image-list (img-size).found ← False
  End If
End For
```

Algorithm (3.17) search on image path to check if image is exists in web site.

Algorithm (3.17) Search-Image

Goal: Check image if really exist in web site or not
Input: img-name
Output: True or False
Variable: path

```
Set path ←Dir(page-name)
If  path ≠ " "  then
   Return true
Else
   Return false
End If
```

Chapter Four

Implementation of Proposed link checker Software

Chapter Four

Implementation of Proposed Link Checker Software

4.1 Introduction

This chapter consists of three parts, the first part explains the programming language used in the proposed link checker software, the second part explain how to run and explain the user interface of the proposed software.

This proposed software work under widows XP operating system. The tests have been applied using a personal computer (Pentium 4, processor 2.66GHz, RAM 256MB).

4.2 Programming Language

Visual Basic was used as programming language for the implementation of the user interface of the system**.**

Visual Basic was used because it is simple**,** easy to learn and provide a flexible user interface to other languages, leaving lots of time for the project design.

4.3 Link Checker Software

The proposed link checker software checking links in Web Site by receiving Web Site's URL as input to extract links and then classified to their types (relative links, fragment links, absolute links and image links).When the user clicks on link checker software icon, the software open the first interface as shown in Figure (4.1) which contain two commands buttons. They are:

1. **Begin Command Button:** when the user clicks on this command button the software going to main menu interface.
2. **Exit Command Button:** this command button is used to exit from the software.

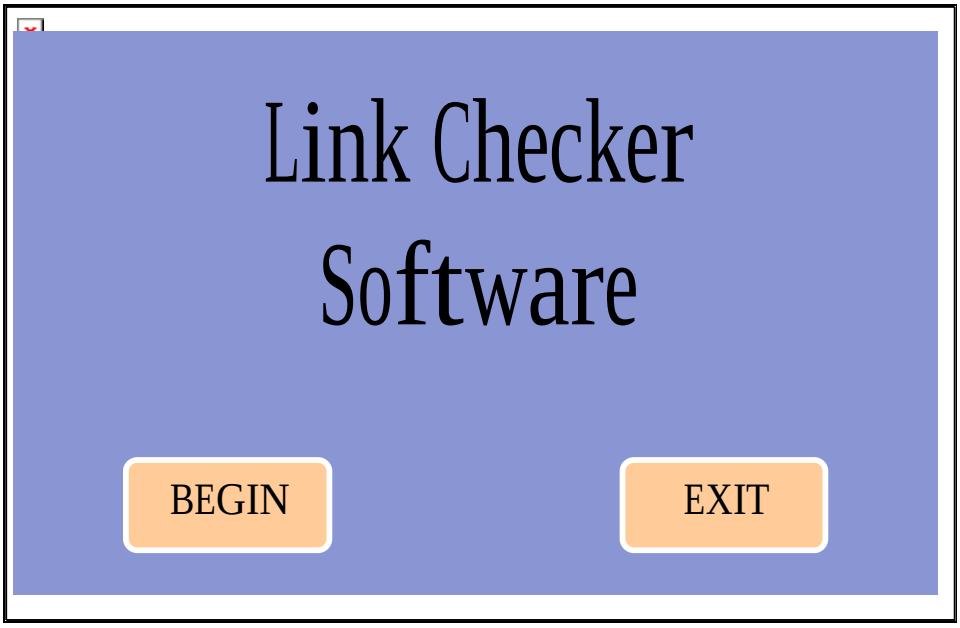


Figure (4.1) Starting interface of proposed software and its Commands

After user clicks on begin command button, the main menu interface appear, the user select one Web Site from the list of Web Sites URLs which found in "Enter URL" field as shown in Figure (4.2) show the Main Menu Interface.

1. **Enter URL Field:** this field is used to select Web Site's URL.
2. **Check Command Button**: this command button is used to extract all links in **a** Web Site and then classified to their types.
3. **Total Web Site Links Command Button**: this command button is used to calculate the number of all links in the Web Site.
4. **Check the Percentage of Good Links Command Button:** this command button is used to check the percentage of good links to Web Site. Where:

Percentage= (no. of good links/no. links of Web Site)*100

Links is representing all type of links.

5. **Check Relative Links Command Button**: this command button is used to display another interface which called relative links checking.
6. **Check Fragment Links Command Button**: this command button is used to display another interface which called fragment links checking.
7. **Check External links Command Button:** this command button is used to display another interface called external Links checking.
8. **Check Image links Command Button:** this command button is used to display another interface called image links checking.
9. **Return Command Button:** this command button is used to return to the starting interface.

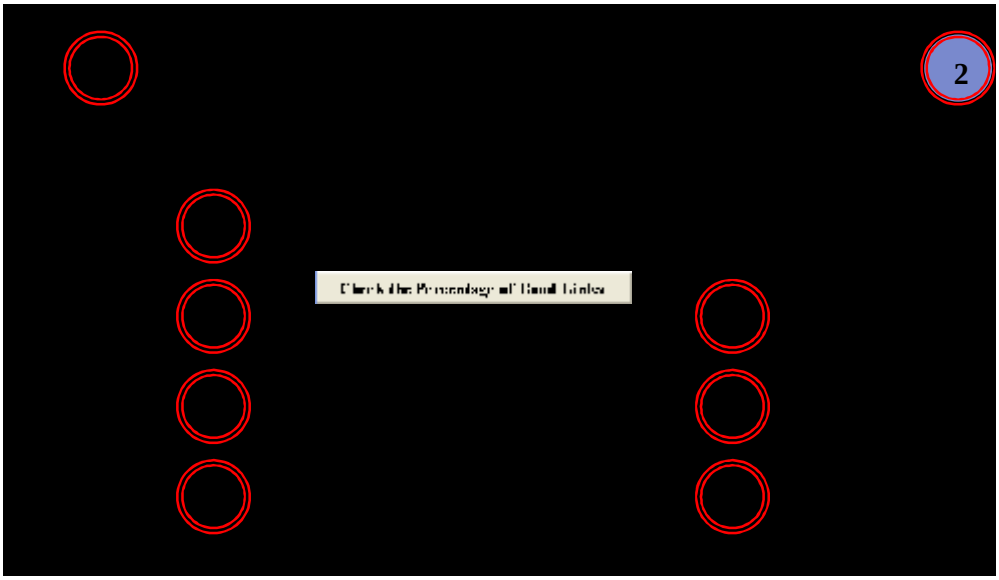


Figure (4.2) Main menu interface

When the user display the main menu interface, at beginning all command buttons (3 to 8) were disabled except command button 9 is enabled. Figure (4.3) show the main menu interface with no user input,

till the user select Web Site's URL, the command button 2 is enabled as shown in Figure (4.4), checking Web Site start when the user click on command button 2, after the user selection the software extracted all links and classified to their types and all command buttons from (3 to 8) were enabled. Figure (4.5) show the main menu interface after user click on command button 2. Each command button (3 to 8) will be discussed in the next sections.

Figure (4.3) Main menu interface with no user input

Figure (4.4) Main menu after user input URL

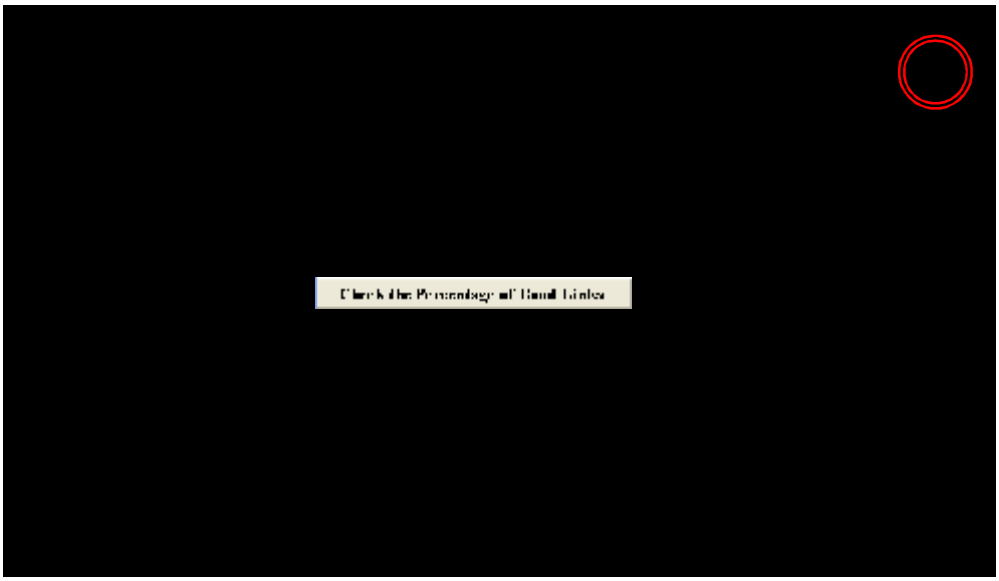


Figure (4.5) Main menu after user click on command button 2

4.4 Relative Links Checking:

When the user click on command button "Check Relative Links" the proposed software opened new interface which called "Relative Links Checking" as shown in figure (4.6), in this interface all in interface information about relative links will be displayed, this interface contains 4 commands buttons to process relative links. These are.

These 4 command buttons in relative links checking interface are:

1. **Check Relative Links Command Button**: when the user click on this Command Button, the software display table of links information. Figure (4.7) show the table of link information.
   a. **No. Field**: it represent index field.
   b. **Source Page Field**: it represents the source link.
   c. **Destination Page Field**: it represents the destination page which the link pointed to.
   d. **Status Field**: it contains two values either GOOD or BAD. If the proposed software find the destination page the status is GOOD else the status is BAD.

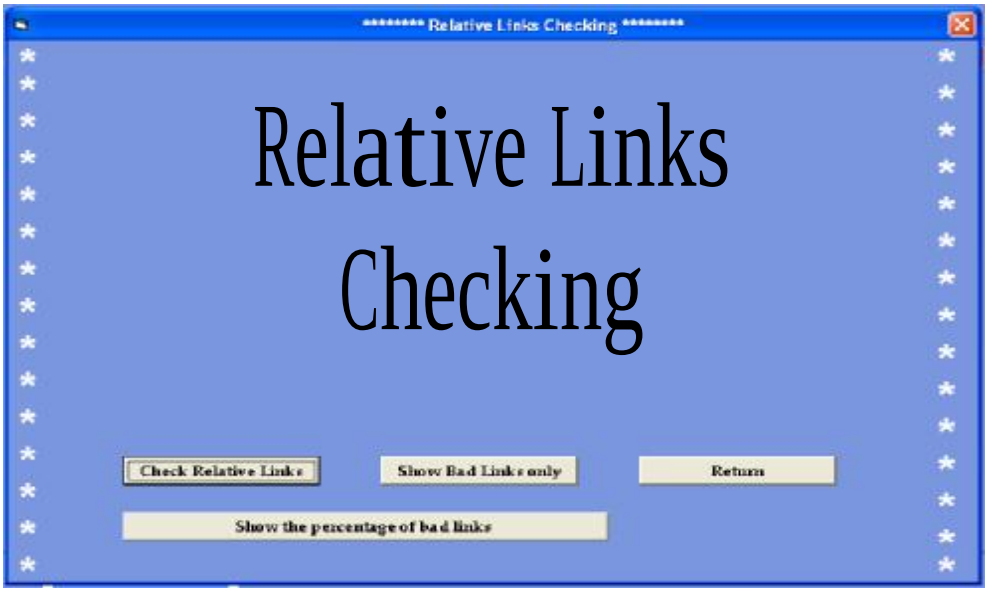


Figure (4.6) Relative links checking interface

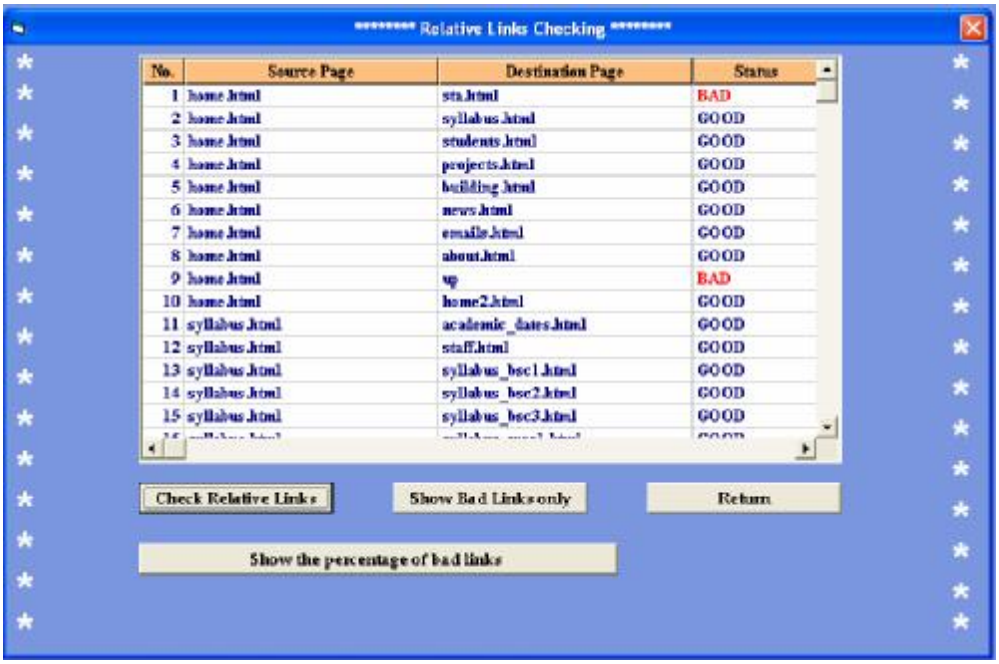


Figure (4.7) Relative links checking interface with information

2. **Show Bad Links Only Command Button**: when the user clicks on this command button the proposed software display bad relative links only. If no bad links in web site, a message will be appeared to user contain this text "No Bad Links". Figure (4.8) show the table of link information with Bad links only.

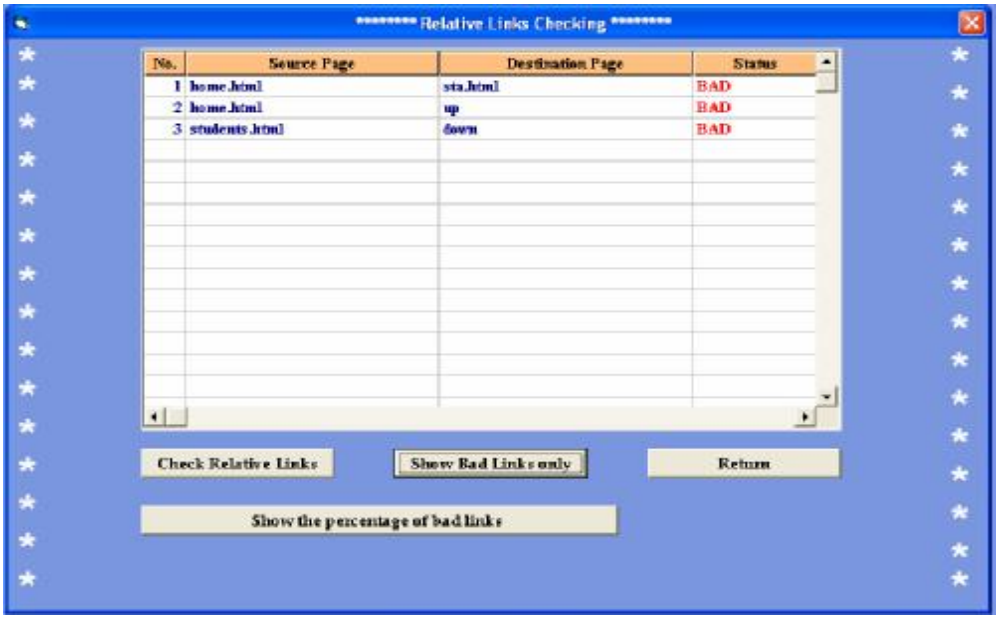


Figure (4.8) Bad links

3. **Show the Percentage of Bad Links Command Button:** when the user click on this command button the percentage of bad links will be calculated. Figure (4.9) show the percentage of bad links. Where: percentage = (no. of bad relative links/total no. of web links)*100
4. **Return**: this command button is used to return to main menu interface.

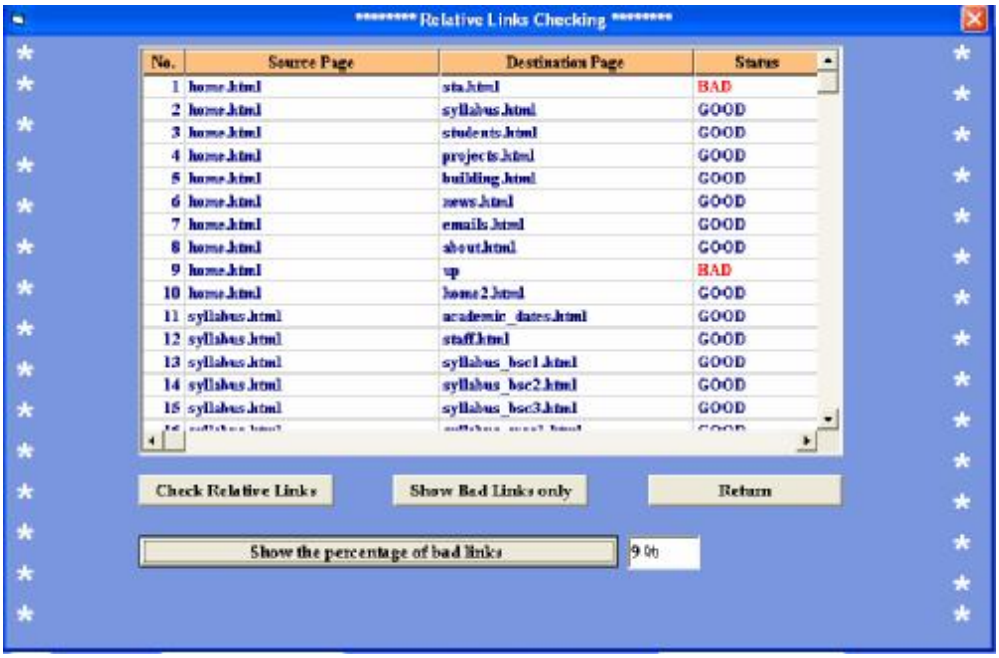


Figure (4.9) Percentage of bad links

4.5 Fragment Links Checking:

When the user click on command button (check fragment links) the software opened new interface which called **fragment links checking**, in this interface all information about fragment links will be displayed, this interface contains 4 command buttons to process fragment links. Figure (4.10) show fragment links checking interface.

These 4 command buttons in fragment links checking interface are:

1. **Check Fragment Links Command Button**: when the user click on this Command Button, the software display table of link information.

Link information is represented by these table's fields. Figure (4.11) show the table of link information.

a. **No. Field**: this field represent index field.
b. **Source Page Field**: this field represents the source link.
c. **Destination Page Field**: this field represents the destination page which the link pointed to it.
d. **Status Field**: this field contains two values either **GOOD** or **BAD**. if the software find the destination page the status is GOOD else the status is BAD.

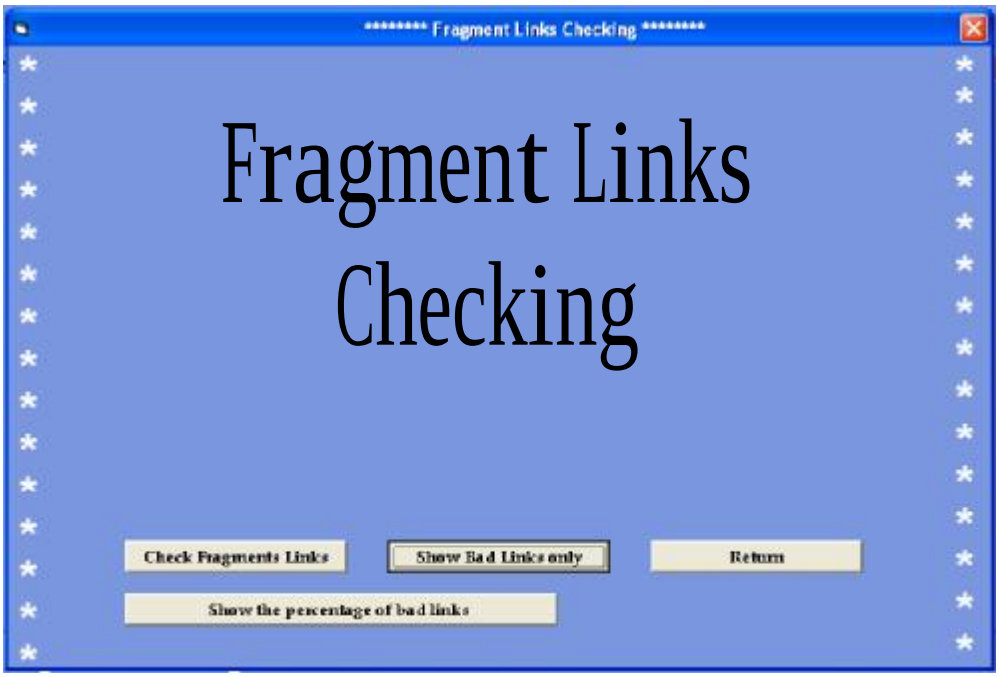


Figure (4.10) Fragment links checking interface

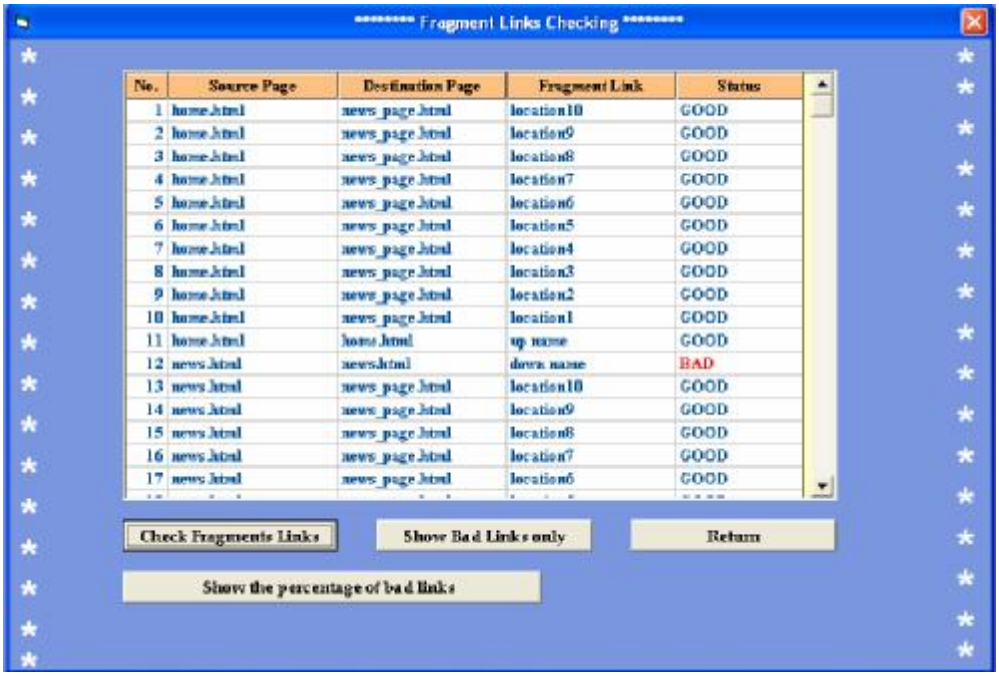


Figure (4.11) Table of link information

2. **Show BAD Links Only Command Button**: when the user clicks on this Command Button the software display bad fragment links only. If no bad links in web site, message will be appearing to user contain this text "No Bad Links". Figure (4.12) show the table of link information with BAD links only.

Figure (4.12) Table of link information with bad links only

3. **Show the Percentage of BAD Links Command Button:** when the user click on this Command Button the percentage of BAD links will be calculated. Figure (4.13) show the percentage of BAD links, where:

Percentage = (no. of bad fragment links/total no. of web links)*100

4. **Return**: this Command Button is used to return to main menu interface.

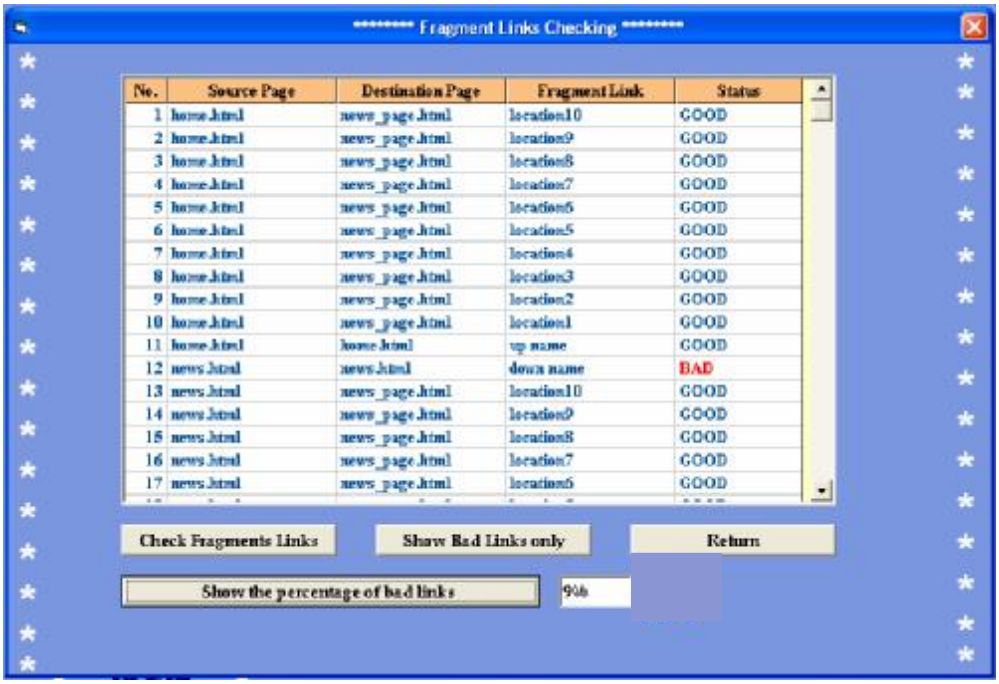


Figure (4.13) Percentage of bad links

4.6 External Links Checking:

When the user click on command button (check external links) the software opened new interface which called **external links checking**, in this interface all information about external links will be displayed, this interface contains 4 Command Buttons to process external links. Figure (4.14) show external links checking interface.

These 4 command buttons in external links checking interface are:

1. Check External Links Command Button: when the user click on this Command Button, the software display table of link information.

Link information is represented by these table's fields. Figure (4.15) show the table of link information.

a. No. Field: this field represent index field.

b. Source Page Field: this field represents the source link.

c. Destination Web Site Field: this field represents the destination page which the link pointed to it.

d. Status Field: this field contains two values either **GOOD** or **BAD**. if the syntax of external links right the status is GOOD else BAD.

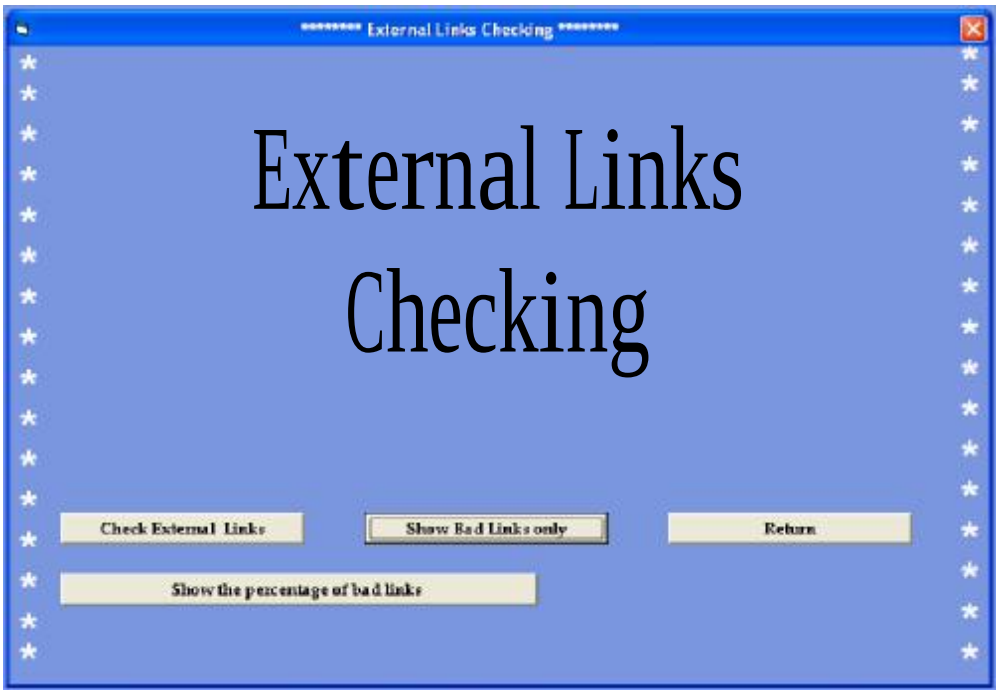


Figure (4.14)External links checking interface

2. Show BAD Links Only Command Button: when the user clicks on this Command Button the software display BAD external links only. If no BAD links in web site, message will be appearing to user contain this text "No BAD Links". Figure (4.16) show the table of link information with BAD links only.

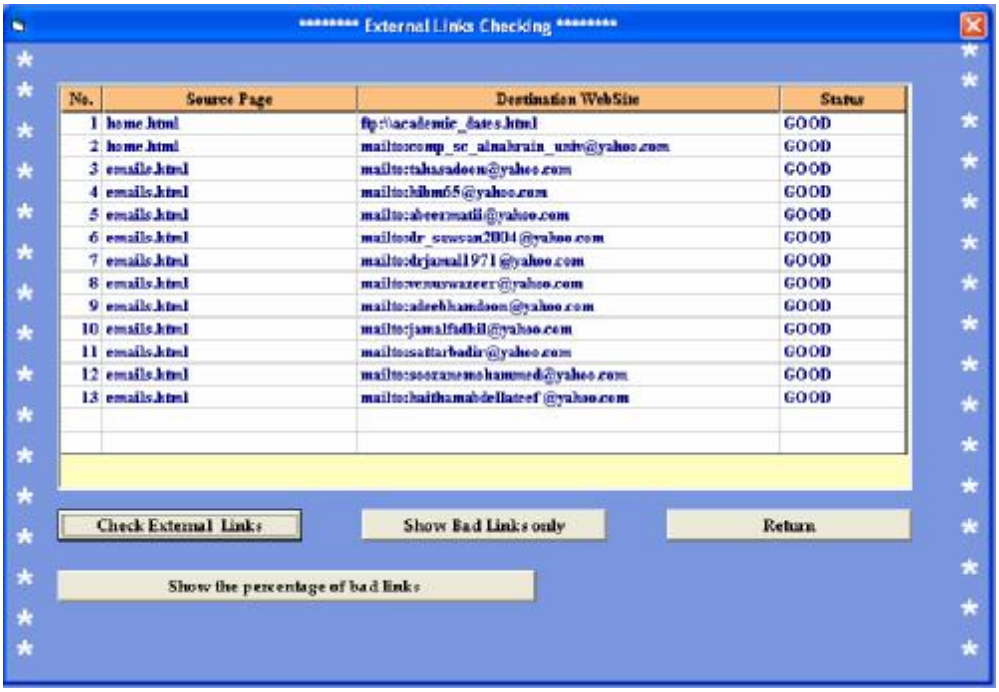


Figure (4.15) Table of link information

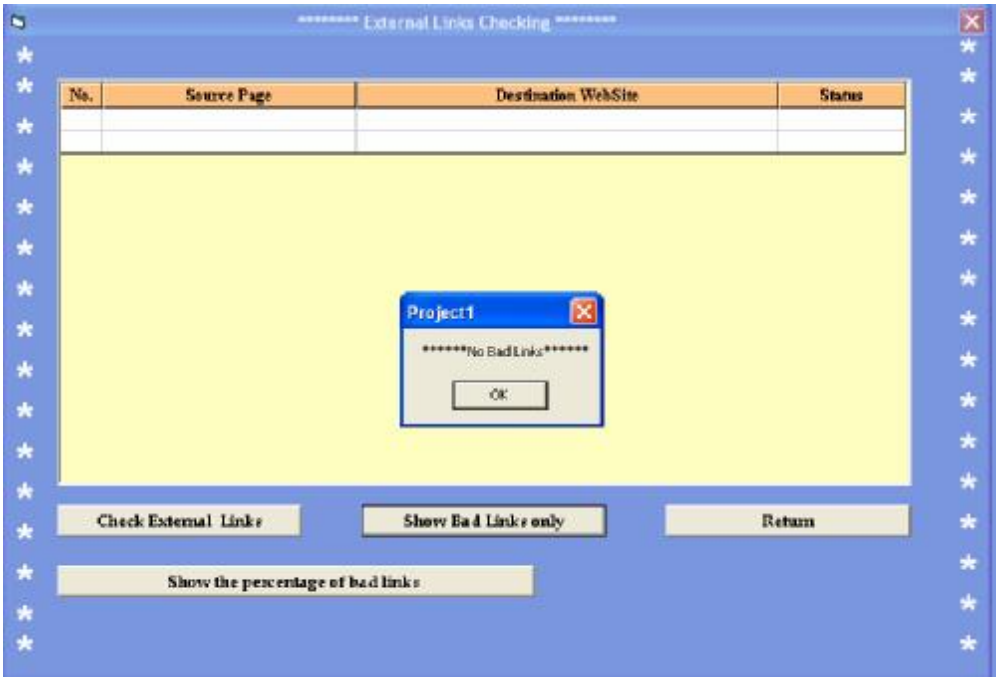


Figure (4.16) Table of link information with bad links only

3. **Show the Percentage of BAD Links Command Button:** when the user click on this Command Button the percentage of BAD links

will be calculated. Figure (4.17) show the percentage of BAD links, where:

Percentage = (no. of bad external links/total no. of web links)*100

4. **Return**: this Command Button is used to return to main menu interface.

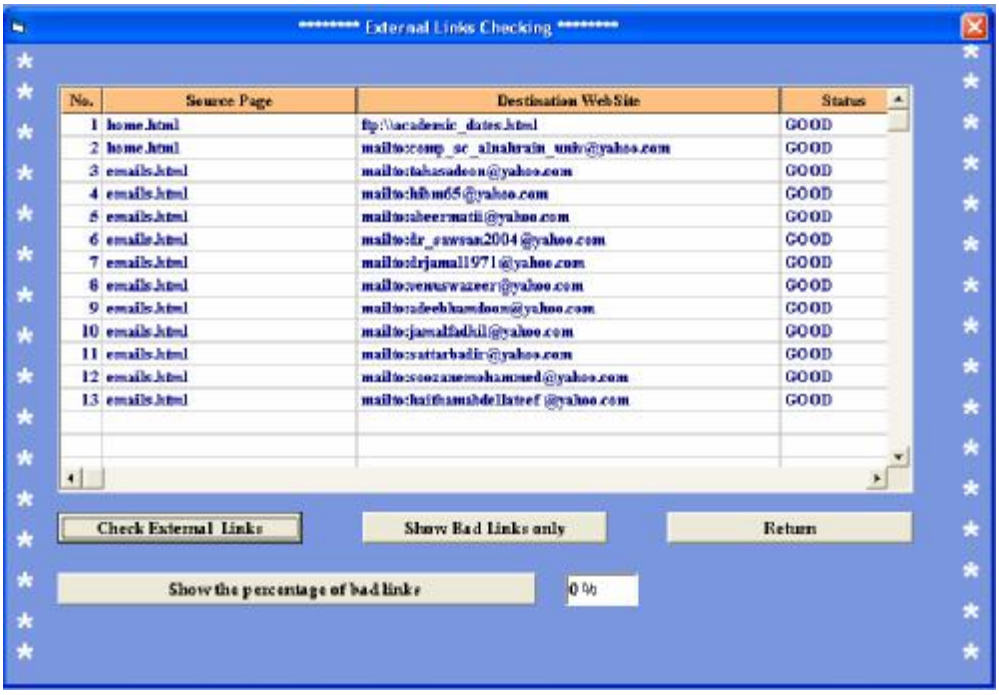


Figure (4.17) Percentage of bad links

4.7 Image Links Checking:

When the user click on command button (check image links) the software opened new interface which called **image links checking**, in this interface all information about image links will be displayed, this interface contains 4 Command Buttons to process image links. Figure (4.18) show image links checking interface.

These 4 command buttons in external links checking interface are:

1. **Check Image Links Command Button**: when the user click on this Command Button, the software display table of link information.

Link information is represented by these table's fields. Figure (4.19) show the table of link information.

a. **No. Field**: this field represent index field.
b. **Source Page Field**: this field represents the source page.
c. **Destination Image Field**: this field represents the destination image which the link pointed to it.
d. **Status Field**: this field contains two values either **GOOD** or **BAD**. if the software find the destination page the status is GOOD else the status is BAD.

Figure (4.18) External links checking interface

2. **Show BAD Links Only Command Button**: when the user clicks on this Command Button the software display BAD image links only. If no BAD links in web site, message will be appearing to user contain this text "No BAD Links". Figure (4.20) show the table of link information with BAD links only.

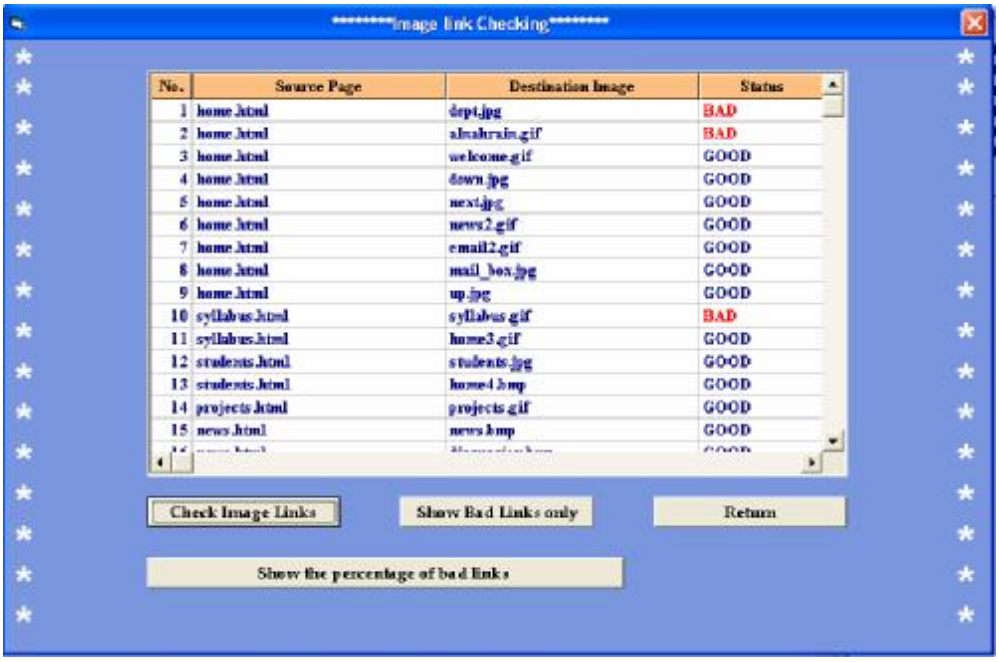


Figure (4.19) Table of link information

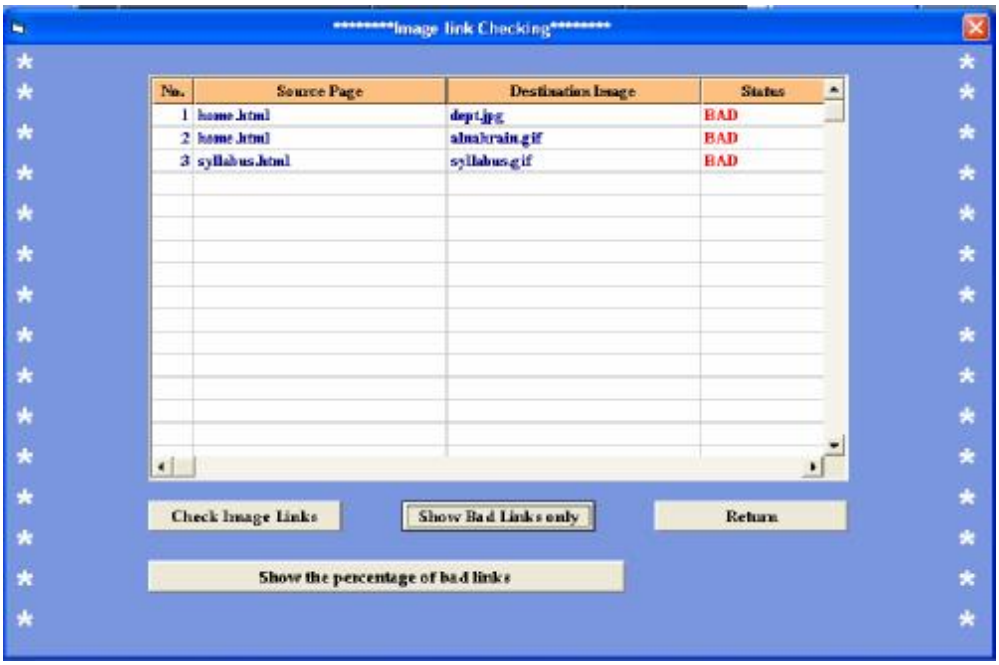


Figure (4.20) Table of link information with BAD links only

3. **Show the Percentage of BAD Links Command Button:** when the user click on this Command Button the percentage of BAD links

will be calculated. Figure (4.21) show the percentage of BAD links, where:

Percentage = (no. of bad image links/total no. of web links)*100

3. **Return**: this Command Button is used to return to main menu interface.

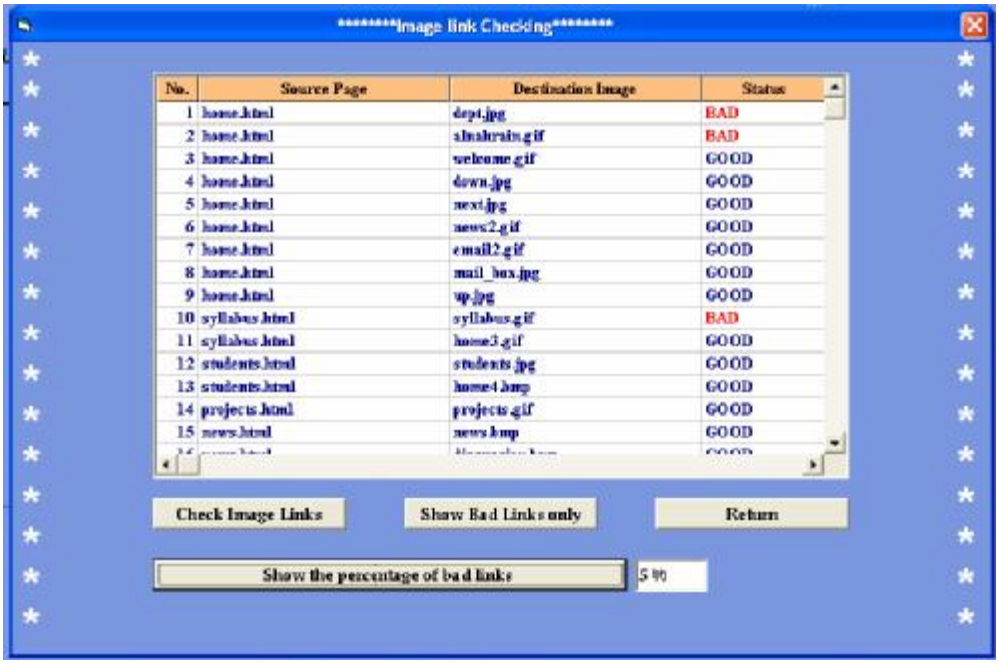


Figure (4.21) Percentage of BAD links

4.8 Evaluation of Proposed Software

After applying proposed link checker software on the computer science department Web Site at al-Nahrain University as sample Web Site links to check its links, it has a total of (130)links :

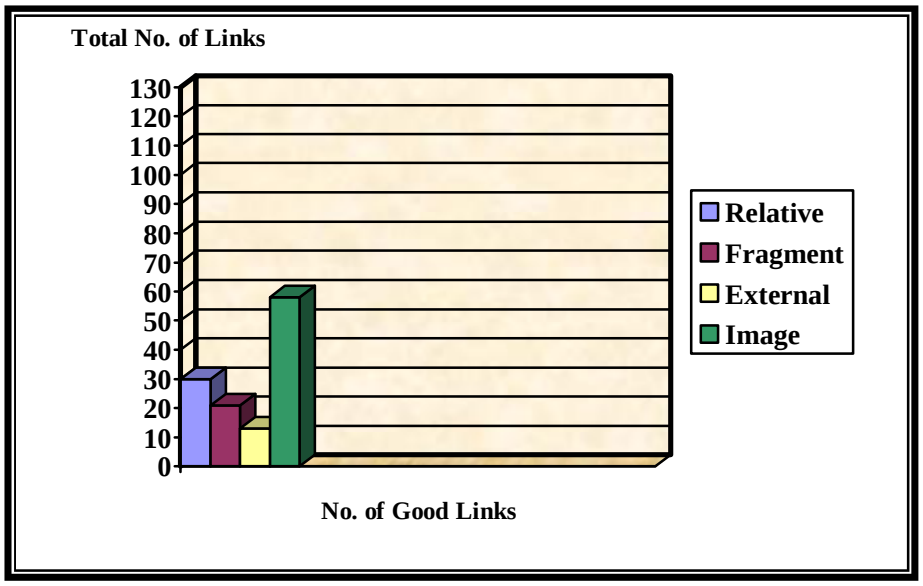
Total No. of Link =130 Links.

Table (4.1) shows the results of applying proposed software on this Web Site.

Link's Type	Total No.	No. of Good Link	No. of Bad Link	Percentage of Bad Link
Relative Link	33	30	3	9%
Fragment Link	23	21	2	8%
External Link	13	13	0	0%
Image Link	61	58	3	5%

Table (4.1) Results of applying proposed software on Web site

This results represented by Flowchart (4.1).



Flowchart (4.1) Results of applying proposed software on Web site

Chapter Five

Conclusions and Suggestions for Future Work

Chapter Five

Conclusions and Suggestions for Future Work

5.1 Conclusions

Through execution and implementation of the proposed link checker software on a Web Site, it is noticed that:

1. Most cases of browsing process in internet shows that there is always a number of bad links in Web sites.
2. The results obtained shows the applicability of the proposed link checker software to find all broken links in HTML Web Site, whether these links are text or image.
3. Some statistical records are evaluated by the software to reflect the efficiency of the Web Site, which is found useful for pre-site evaluation and Web Site developers.
4. The presented link checker can be used as a useful tool after off line maintenance of any Web site, and before uploading the Web Site for production.

5.2 Suggestions for Future Work

1. Construct more flexible system which works on Web site written by PHP language in addition to HTML language.
2. Developing the current software to support audio and video links.
3. Connect proposed software to online for the check of external links.

References

REFERENCES

[Ann99] Ann N. and Todd S.,"HTML by Example", Que, Inc., 1999

[Avi99] Avi R., "Robots and Spiders and Crawlers", 1999, available at

[Bil00] Bill G., "404 Error Message", 2000, available at http://www.get-articales.com

[Chu98] Chuck M. and Bill, K., "HTML the Definitive Guide ", 3rd Edition, O'Reilly and Associates, Inc., 1998

[Dav00] David M.,"CGI programming with Tcl", Addison-Wesley, 2000

[Eih05] Eihab A., "Development of a Web Site Search Engine", M.Sc. Thesis, Al-Nahrain University, 2005

[Gre99] Greg R., "Raising Deal links", Reference Librarian Montana State University, Online Inc., 1999

[Hea05] Heather M., "Non- usability Testing methods for Learning how Web Sites function", Technical communication ", Volume 52, 2005

[Hon02] Hongjing W., "A Reference Architecture for Adaptive Hypermedia Applications", University Press Facilities, Eindhoven, the Netherlands, 2002

[Ing95] Ingham D., B., Canghey S. J. and Litlle M.C., "Fixing the Broken Link Problem", United Kingdom, 1995

[Jef01] Jeffery V., "The Art and Science of Web Design", 2nd, United state of America, 2001

[Jen99] Jenneifer N., "Web Design in a Nutshell", 1st Edition, Orally and Associates Inc., 1999

[JHL98] Cho J. , Garcia-Molina H., Page L., "Efficient Crawling Through URL Ordering", Department of Computer Science, Stanford University, 1998

[Jim03] Jim B., "Microsoft Office FrontPage 2003 Inside Out", Microsoft Press, 2003

[Lin02] Linktek Corporation, "Link Fixer Plus", available at www.linkfierplus.com, 2002

[Mic02] Michael D. Larue, "link Controller", General Public License, 2004

[Pau04] Paulo M., "Klinkstatus", General Public License, 2004

[Rya00] Ryan, H., "How Dead are Dead Links", 2000

[Sab02] Saba A., "Internet and Arabic Search Engines", M. Sc. Thesis, Al-Nahrain University, 2002

[SL98] Birn S. and Page L., " The Anatomy of a Large Scale Hypertextual Web Search Engine", Computer Science Department, Stanford University, 1998

[Sun01] Sunny L., "The Overview of Web Search Engines", Department of Computer Science, University of Waterloo, Ontario Canada, 2001

[Vir01] Virtualis Glossary, "Virtualis System", 2001, available at http://www.virtualis.com/guides-glossary.html

[W3c04] W3c Recommendation," Architecture of the World Wide Web", Volume one, 2004

[Zub03] Zubin J., "Solving the Broken Link Problem in Walden's Paths", M.Sc. Thesis, Texas A and M University, 2003

الخلاصـــه

ان مشكلة الروابط المقطوعه في المواقع الالكترونيه هي من المشاكل الشائعه علـــى شـــبكة الانترنيت في الوقت الحاضر.وتؤدي هذه المشكله الى نفور زوار الموقع الالكتروني وتمنع عمليـــة التصفح داخل الموقع بصوره سليمه والوصول الى المعلومات المطلوبه,وبما ان مجتمعنا اصبح اكثر اعتماداً على الانترنيت فأن من اهم الاولويات هي ضمان عمل الروابط بشكل صحيح.

ان هذا البحث يهدف الى تصميم وتنفيذ نظام يقوم باختبار جميع روابط الموقع الالكتـــروني وبعدها يقوم النظام بتصنيفها الى اربع قوائم طبقاً الى نوع الرابط(قائمة الـــرابط النـــسبي،قائمة الرابط الضمني،قائمة الرابط المطلق و قائمة الرابط الصوري)وبعد عملية التصنيف يقوم النظـــام بعملية تدقيق الروابط في كل قائمه لكي يتحقق فيما اذا كان الرابط يعمل بشكل صحيح او كلا واعطاء تقرير حول جميع الروابط.

ان هيكلية نظام مدقق الرابط المقترح يتألف من ثلاث وحدات رئيسيه,الاولى وتدعى وحدة المتابعه و استخراج الرابط النصي, والثانيه تدعى وحدة استخراج الرابط الصوري,والثالثه تدعى وحدة التدقيق والتي تتكون من وحدتين فرعيه وهما وحدة تدقيق الرابط النصي و وحدة تدقيق الرابط الصوري.

ان مدقق الرابط المقترح يطبق على المواقع الالكترونيه المكتوبه بلغة ال(HTML).ان مدقق الرابط اعتمد الموقع الالكتروني لجامعة النهرين كنموذج للاختبار وقد تبين من النتائج المستحصله من تطبيق البرنامج المقترح قدرة مدقق الرابط على اكتشاف جميع الروابط المقطوعه في الموقع الالكتروني. كذلك احتسبت بعض الاحصائيات المسجله من قبل البرنامج والتي وجدت بانها مفيده للتقييم مسبقاً و لمطورين الموقع الالكتروني.

ان مدقق الرابط يستعمل كأداة مهمه في عملية الأدامه وخاصةً كخطوه اخيره قبل رفعه للنشر على الانترنيت.

لقد تم في هذا النظام المقترح استخدام الادوات البرمجيه التاليه: Microsoft Visual Basic 6.0, Hypertext Markup Language and Windows operating system.

جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة النهرين
كلية العلوم

تطبيق مدقق الرابط لتحليل الموقع الالكتروني

رسالة

مقدمه الى كلية العلوم في جامعة النهرين كجزء من متطلبات نيل درجة الماجستير في علوم الحاسبات

من قبل

إحسان قحطان أحمد

(بكالوريوس جامعة النهرين 2002)

إشراف

د.عبد المنعم صالح رحمه

د.جمال فاضل توفيق

تشرين الاول 2009

شوال 1430