An environment for the development and modification of dynamic hypertext information structures.

May. Wang
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/1000

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.
An Environment for the Development and Modification of Dynamic Hypertext Information Structures

by

May Wang

A Thesis
Submitted to the Faculty of Graduate Studies and Research through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the University of Windsor

Windsor, Ontario, Canada
1995
The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

### THE HUMANITIES AND SOCIAL SCIENCES

#### COMMUNICATIONS AND THE ARTS
- Architecture: 0729
- Art History: 0737
- Cinema: 0700
- Dance: 0770
- Fine Arts: 0720
- Information Science: 0723
- Journalism: 0731
- Library Science: 0739
- Mass Communications: 0728
- Music: 0740
- Speech Communication: 0749
- Theater: 0745

#### EDUCATION
- General: 0515
- Administration: 0516
- Adult and Continuing: 0517
- Agricultural: 0518
- Adult: 0723
- Bilingual and Multicultural: 0521
- Business: 0588
- Community College: 0522
- Curriculum and Instruction: 0727
- Early Childhood: 0516
- Elementary: 0524
- Finance: 0777
- Guidance and Counseling: 0519
- Health: 0582
- Higher: 0745
- History: 0725
- Home Economics: 0526
- Industrial: 0521
- Language and Literature: 0529
- Mathematics: 0527
- Music: 0523
- Philosophy of: 0523
- Physical: 0523

#### THE SCIENCES AND ENGINEERING

#### BIOLOGICAL SCIENCES
- Agriculture: 0473
- Agronomy: 0475
- Animal Science: 0475
- Nurture: 0475
- Animal Production: 0475
- Forestry and Wildlife: 0359
- Plant Science: 0479
- Plant Pathology: 0480
- Plant Physiology: 0478
- Range Management: 0772
- Wood Technology: 0746
- Biology: 0306
- Anatomy: 0319
- Biochemistry: 0306
- Botany: 0306
- Cell: 0306
- Ecology: 0306
- Endocrinology: 0306
- Genetics: 0306
- Limnology: 0306
- Microbiology: 0410
- Molecular Biology: 0410
- Neurobiology: 0410
- Oceanography: 0416
- Physiology: 0416
- Radiobiology: 0416
- Radiation Biology: 0416
- Zoology: 0416

#### PSYCHOLOGY
- Biocology: 0416
- Geochronology: 0416
- Neurosience: 0416
- Optics: 0416
- Zoology: 0416

### HEALTH AND ENVIRONMENTAL SCIENCES
- Environmental Sciences: 0748
- Health Sciences: 0566
- General: 0566
- Audiology: 0566
- Biochemistry: 0566
- Biophysics: 0566
- Ecology: 0566
- Endocrinology: 0566
- Environmental Science: 0566
- Immunology: 0566
- Medical Science: 0566
- Nutrition: 0566
- Occupational Therapy: 0566
- Ophthalmology: 0566
- Oncology: 0566
- Occupational Health: 0566
- Therapy: 0566

### PHYSICAL SCIENCES
- Pure Sciences: 0475
- Chemistry: 0475
- General: 0475
- Agricultural: 0475
- Analytical: 0475
- Biochemistry: 0475
- Inorganic: 0475
- Organic: 0475
- Physical: 0475
- Radiation: 0475
- Mathematics: 0475
- General: 0475
- Acoustics: 0475
- Astronomy and Astrophysics: 0475
- Atmospheric Science: 0475
- Atomic: 0475
- Electronics: 0475
- Environmental Science: 0475

### ENGINEERING
- General: 0457
- Aerospace: 0538
- Agricultural: 0539
- Automotive: 0540
- Biomedical: 0541
- Chemical: 0542
- Civil: 0543
- Computer Science: 0544
- Engineering: 0545
- Marine: 0546
- Materials Science: 0547
- Mechanical: 0548
- Metallurgy: 0549
- Mining: 0549
- Nuclear: 0550
- Packaging: 0550
- Petroleum: 0550
- Sanitary and Municipal: 0550
- System Science: 0550
- Geology: 0550
- Operations Research: 0550
- Plastic Technology: 0550
- Textile Technology: 0550

### PSYCHOLOGY
- General: 0621
- Behavioral: 0621
- Clinical: 0622
- Developmental: 0620
- Experimental: 0623
- Industrial: 0624
- Personality: 0625
- Physiological: 0626
- Psychometric: 0627
- Psychometrics: 0628
- Social: 0629
May Wang 1995
© All Rights Reserved
Abstract

The World-Wide Web (WWW) changes the way people retrieve information from the Internet. The Web is regarded as a public information system which offers access to the network community with the client/server protocol Hypertext Transfer Protocol (HTTP). The Web uses client interface programs (browsers), hypertext and multimedia techniques to make it easy for anyone to roam, browse, and contribute to the information base. However, most information available on the Web is read-only, frozen messages. The introduction of HTML Fill-out Forms and Common Gateway Interface changes this situation.

In this thesis we explore supplying dynamic information to users through the Web. We build an environment on the Web using CGI and HTML forms. This environment uses forms to let the user interact with the Web server, and generates text, pages and hyper links on the Web based on the users’ requirements. This environment is accessible to all authorized users and the information in it is changeable. All changes are visible immediately to all users. Applications of the environment are described.
To my dearest husband Phillip Jia
my mother Yunling Mei
my father Jiangong Wang
my grandmother Guiying Xu and
my grandfather Zhongkui Xu
Acknowledgments

I would like to acknowledge the support and guidance provided by Dr. R. Frost, whose time, patience, support and effort have contributed in guiding me through my master study and this thesis. I am grateful to Dr. D. Kao and Dr. J. Morrissey for providing valuable suggestions and comments that have aided in my progress. I would also like to thank Mr. S. Karamatos and Mr. W. Mnaymneh for their help. I express my gratitude to Ms. E. Preuschat for making corrections in this thesis.

Special thanks to my husband for always being there for me. His constant support and encouragement have contributed significantly towards my successful completion.
Contents

Abstract ................................................................. iv
Acknowledgments ......................................................... vi
List of Figures ............................................................ xi
List of Tables ............................................................. xii

1 INTRODUCTION ......................................................... 1
  1.1 The Internet and World-Wide Web ............................... 1
  1.2 The Features of the World-Wide Web ......................... 2
    1.2.1 Universal Readership and Hypermedia Documents ...... 2
    1.2.2 MIME Typing ................................................. 3
    1.2.3 Client/Server Model ....................................... 5
    1.2.4 Browsers .................................................... 6
    1.2.5 Hypertext Transfer Protocol (HTTP) ....................... 6
    1.2.6 Uniform Resource Locators (URL) ......................... 7
  1.3 Overview of the Thesis ......................................... 9
    1.3.1 Motivation ................................................. 9
    1.3.2 Thesis Statement .......................................... 9
    1.3.3 Objective of the Thesis .................................. 10
    1.3.4 Organization of the Thesis ............................... 10

2 Fill-Out Forms and the Common Gateway Interface ................ 12
  2.1 Fill-out Forms ................................................ 12
    2.1.1 The Basic Structure of a Form .......................... 12
    2.1.2 Inputting the Information ............................... 14
2.1.3 Building a Form .............................................. 15

2.2 Common Gateway Interface .................................. 15

2.2.1 Programming With CGI .................................. 15

2.2.2 Handling HTML Forms ................................. 16

3  An Environment for the Development and Modification of Dynamic
   Hypertext Information Structures .............................. 18

3.1 System Design ............................................. 19

3.2 System Development ....................................... 21

3.2.1 The Log-on Page ........................................ 21

3.2.2 Pages of the Dynamic Hypertext Information Structure Builder .......................... 21

3.2.2.1 Creating a Page ..................................... 23

3.2.2.2 Opening a Page ..................................... 25

3.2.3 Creating a Hyperlink to Anywhere On the Web ......... 25

3.3 Text File Manipulation in the System ........................ 27

3.3.1 Putting a Text on the Web through our Environment .......... 27

3.3.2 Locking Facilities for Multiple Readers and Single Writer .......................... 31

3.4 Implementation Problem: Browser Caching .................. 32

4  A Dynamic Hypertext Information Structure Development Using Our
   Environment .................................................. 36

4.1 Home page .................................................. 36

4.2 Bulletin Board ............................................. 37

4.3 Dynamic Information Structure ................................ 39

4.3.1 Examples of Application Development ..................... 40
4.3.2 Building A Dynamic Hypertext Information Structure for Faculty

Members of the School of Computer Science ............ 41

5 Conclusions and Future work ........................................ 47
5.1 Conclusions ......................................................... 47
5.2 Future Work ....................................................... 48

Appendix A HTML Code of the Log-on Form ......................... 50
Appendix B Function Library for CGI Scripts ....................... 51
Appendix C CGI Script for Log-on Checking ......................... 55
Appendix D Form for Creating New Page ............................... 57
Appendix E CGI Script for Creating New Page ....................... 58
Appendix F CGI Script for Opening A Page ......................... 60
Appendix G CGI Script for Deleting A Page ......................... 61
Appendix H Form for Creating New Hyper-Link .................... 63
Appendix I CGI Script for Creating New Hyper-Link ............... 64
Appendix J Form for Adding New Text ............................. 67
Appendix K CGI Script for Adding New Text ....................... 68
Appendix L Form for Appending Messages ......................... 69
Appendix M CGI Script for Appending Messages ................... 70
Appendix N Form for Editing Text .................................. 73
Appendix O CGI Script for Editing Text ............................ 76
Appendix P Form for Deleting Text ................................ 79
Appendix Q CGI Script for Deleting Text ........................... 81
Appendix R  CGI Script for Creating Index Tree  ................. 82

BIBLIOGRAPHY  .................................................. 85

Appendix S  VITA AUCTORIS  ........................................ 86
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic Hypertext method</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Client/Server Model</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>A simple URL diagram</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>A log-on form on the Web</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>Example Page</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>Form for creating a new page</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>Form for creating hyperlink</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>Form for putting new text on the Web</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>Form for appending a response to the text item</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>Example page</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>Dynamic page directory tree</td>
<td>34</td>
</tr>
<tr>
<td>12</td>
<td>Home page</td>
<td>36</td>
</tr>
<tr>
<td>13</td>
<td>Bulletin board</td>
<td>37</td>
</tr>
<tr>
<td>14</td>
<td>An information item on the Bulletin Board</td>
<td>39</td>
</tr>
<tr>
<td>15</td>
<td>Top page of the environment</td>
<td>40</td>
</tr>
<tr>
<td>16</td>
<td>Faculty members administrative tasks</td>
<td>41</td>
</tr>
<tr>
<td>17</td>
<td>Form for creating new page</td>
<td>42</td>
</tr>
<tr>
<td>18</td>
<td>A page</td>
<td>43</td>
</tr>
<tr>
<td>19</td>
<td>Faculty members administrative tasks page</td>
<td>44</td>
</tr>
<tr>
<td>20</td>
<td>A page with text and hyper-link</td>
<td>46</td>
</tr>
</tbody>
</table>
List of Tables

Table 1  | Text types commonly found on the Web  | 4
---------|--------------------------------------|----
Table 2  | Image types commonly found on the Web | 5
Table 3  | Sound types commonly found on the Web | 5
Chapter 1 INTRODUCTION

1.1 The Internet and World-Wide Web

The major developments in distributed computing from the 60's to the 90's are characterized by the ARPAnet national research network which merged with subsequent networks to form the world-wide Internet. The Internet is the biggest network the world has ever seen. It offers public access to the network community. As the popularity of Internet increased, people become more aware of its colossal potential. The World-Wide Web (WWW) is a product of the continuous search for innovative ways of sharing resources through the Internet [7] [5]. From a structural point of view, the Web is made up of four parts, as follows:

- Hypermedia Documents
- Hypertext Transfer Protocol (HTTP)
- Uniform Resource Locator (URL)
- Browsers

The Web runs on the Internet, and can't exist without the Internet.

WWW was originally developed to allow information sharing within internationally distributed teams, and the dissemination of information by support groups. Originally aimed at the High Energy Physics Community, it has spread to other areas and attracted much interest in user support, resource discovery and collaborative work areas. The Web is currently the most advanced information system deployed on the Internet, and embraces within its data model much information in previous networked information systems. In fact, the Web is an architecture which will also embrace many future advances in technology, including new networks, protocols, object types and data formats. The
World-Wide Web consortium (W3C) guides the growth of the Web. The goal of the
W3C is to develop the Web into a global information infrastructure capable of supporting
commercial as well as research activities.

1.2 The Features of the World-Wide Web

The World Wide Web is the universe of network-accessible information. It has a
body of software, and a set of protocols and conventions. WWW uses hypertext and
multimedia techniques to make the Web easy for anyone to surf, browse, and contribute
to. The Web is considered a seamless world in which all information, from any source,
can be accessed in a consistent and simple way.

Universal Readership and Hypermedia Documents

Before WWW, one had to have one of a number of different terminals connected to
a number of different computers, and one had to learn a number of different programs to
access that data. The WWW principal of universal readership is that once information
is available, it should be accessible from any type of computer, in any place, and an
authorized person should only have to use one simple program to access it. In practice,
the WWW uses the concept of hypertext.

Hypertext is text with links. The term, invented by Ted Nelson in 1965, usually means
text that is not constrained to be linear. Hypertext escapes the bounds of linearity by
means of links or references to other texts. So the reader can escape from the sequential
organization of the pages to pursue a thread of his or her own. Hypertext authors design
their material to make it open to active exploration, and in doing so communicate their
information and ideas more effectively. The following figure shows the hypertext method.
WWW uses hypertext as the method of presentation, although this does not necessarily require that authors write hypertext. In WWW, links can lead from all or part of a document to all or part of another document. Documents need not be text; they can be graphics, movies and sound, so the term "hypermedia", meaning "multimedia hypertext" can be applied equally well to WWW.

**MIME Typing**

One of the most impressive aspects of the World-Wide Web is how transparent the transmission of multimedia information can seem. A single link can bring up a text file, a document from across the world, a movie or a recording. The reason for this seeming transparency is that the server, when sending a file of any type, can tell the client how to process that file. For example, the server tells the client whether to run the file through an audio tool or an animation program or to simply load it as part of a new Web page.
To tell the client how to process the file, the server sends the file with an attachment specifying its type. This attachment is referred to as the file’s *MIME type*, which stands for Multipurpose Internet Mail Extensions.

This name comes from the original purpose of MIME typing, which was to allow for multimedia components to be sent with electronic mail messages by specifying their types in advance so that the mail program could display them as part of the message. MIME typing tells the browser what kind of file is being sent, so that the browser can then tell the client’s computer how to display it. MIME types such as x-compressed, x-gzp, x-tar, and x-zip-compressed are understood to be compressed file types, and a browser serves files of these types to the client’s drive.

Table 1 is the text types commonly found on the Web.

<table>
<thead>
<tr>
<th>Name</th>
<th>Extension(s)</th>
<th>MIME Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyperText Markup</td>
<td>.html, .htm</td>
<td>text/html</td>
<td>The basis for the WWW</td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>.txt</td>
<td>text/plain</td>
<td>Plain ASCII text</td>
</tr>
<tr>
<td>Rich Text Format</td>
<td>.rtf</td>
<td>application/rtf</td>
<td>An interchangeable text file type that preserves some formatting</td>
</tr>
<tr>
<td>PostScript</td>
<td>.ps, .ai, .eps</td>
<td>application/postscript</td>
<td>PostScript files</td>
</tr>
</tbody>
</table>

Table 2 is the image types commonly found on the Web.
Table 2 Image types commonly found on the Web

Table 3 is the sound types commonly found on the Web.

Table 3 Sound types commonly found on the Web

**Client/Server Model**

To allow the Web to scale, it was designed without any centralized facility. Anyone can publish and read information. There is no central control. To publish data you run a server, and to read data you run a client. All the clients and all the servers are connected to each other by the Internet. The following figure shows the Client/Server mode of WWW.
Browsers

A browser is the client in the Web's Client/Server model. A client program is one that you interact with. The server, which you don't touch, supplies information to the client as it relays your requests to it. For a user, a browser is the gateway to the Web. Everything that you see on the Web pages through the lens of your browser. A browser can only show the files that it understands. Most files will be hypermedia files written in Hypertext Markup Language (HTML), and any Web browser will be able to read them.

Hypertext Transfer Protocol (HTTP)

Since computers were invented, there has been a great variety of different codes for representing information. It has never been possible to pick one as the "best" code, as each has its advantages and its advocates.
Hypertext Transfer Protocol (HTTP) is a simple data transfer protocol that binds the Web together [4]. It is based on a Client/Server model which uses a stateless protocol. Essentially, the protocol consists of a set of messages and replies for both servers and browsers [2]. In HTTP, documents, files, menus, and graphics are all treated as objects. A feature of HTTP is that the client sends a list of the representations it understands along with its request, and the server can then ensure that it replies in a suitable way. Thus the WWW can cope with the existing mass of graphics formats like Graphics Interchange Format GIF, TIFF, Joint Experts Group JPEG, etc.. HTTP relies on the Uniform Resource Identifier (URL) to identify files. Although it uses the Internet’s TCP/IP network protocol, HTTP isn’t limited to TCP/IP. HTTP can be implemented to run on other network protocols such as NetWare’s Internet Packed Exchange IPX.

HTTP runs on Web servers. A Web server is software that, upon receiving a browser request, sends the requested document back to the browser. If for some reason it cannot send the file, the server sends a simple error message.

Uniform Resource Locators (URL)

Uniform Resource Locators are the addresses of Web resources. Usually, a URL leads to a file, but it can also point you to a single record in a database, the front-end of an Internet program such as Gopher, or the results of a query. The following figure shows a simple URL.
The first session (http:) tells you the type of resource you’re connecting to. In this case, it’s an HTTP resource. The double slashes (//) indicate that you’re talking to a server.

The next component of the URL is the Internet domain name and address of the web server (www.cs.uwindsor.ca). When you have a complete address such as this, it’s called a fully qualified domain name. The letters at the end of the address, depending on the naming style used by the site, either tells you something about who owns the server or where the server is located. This is similar to an E-mail address.

The next component is the path to the destination. At the end of the URL is the actual resource or resource name which can be a file with a file extension name. Most of the time, users don’t have to type the URLs because they can usually just click on the link.

The address works in the following manner: The whole network is held together by links. The browser scampers over the strands of TCP/IP with a TCP request to a URL. At the other end of the URL sits an HTTP server that sends the requested information back, using HTTP. If all goes well, this request is received by the server’s HTTP program. Normally, HTTP programs run as daemons — programs that are always running in the background and are constantly checking to see if they need to perform their job. After the HTTP daemon receives your request, it relays it to the appropriate service.
When the service required is to fetch an HTML document, the HTTP daemon fetches the document and sends it to the client using HTTP. If it’s some other resource that HTTP can access directly, it still follows this routine. HTTP works differently when users call upon other Internet services such as FTP or Gopher. In these cases, HTTP acts as a go-between. For instance, when a user transfers a file with HTTP, the HTTP server logs them in with a login id of ‘anonymous’ and uses the user’s Internet address as a password.

1.3 Overview of the Thesis

Motivation

The World-Wide Web, in conjunction with its client browser, is an extremely effective mechanism for individuals to share distributed information. However, most of the Web documents are read-only, frozen information. Access to such documents is unidirectional, and asynchronous. This is because the HTTP servers use the Hyper Text Transfer Protocol (HTTP) which is designed to be a stateless protocol. The one-directional request mechanism constrains the HTTP server to become a passive participant in any transaction between the user and the browser. This mechanism might work well when displaying static hypertext documents, but it is not sufficient when significant interaction in both directions is required.

The work done in this thesis is to build an environment to help a group of people create dynamic hypertext information structures on the Web through a client browser.

Thesis Statement

This thesis defends the following statements:

1. A Web browser can be used for the interactive communication between the front-end user and the Web.
2. The Web can supply dynamic information pages instead of read-only, frozen messages.

3. The Web can be used to support a cooperative work environment.

Objective of the Thesis

The objective of this thesis is to improve the interactive communication between the Web server and the front user. In particular, it focuses on supplying dynamic information structures to the users which is different to most read-only, frozen Web documents. People can access the dynamic structures with a standard W3 browser and every member of an authorized group can manipulate all information in the system; thus the whole group can work cooperatively on the same project on the Web. The system is designed to be easy to use. All information in the system can be inspected and changed with a uniform point and click interface. It is a read-write system; the information in it can be changed and added to interactively, and all changes and additions are visible immediately to all users. Thus the system is not just a place for storing information; it is also a medium for communication among its users.

Organization of the Thesis

This thesis is organized into five chapters. Chapter 1 gives a brief introduction of the World-Wide Web, and an overview of the thesis.

Chapter 2 explains, in detail, HTML Fill-out Forms and the Common Gateway Interface.

Chapter 3 describes the design and implementation of the environment for the development and modification of dynamic hypertext information structures.
Chapter 4 describes the environment's user-friendly interface and present some applications of our environment in developing a dynamic information system for faculty administrative work of the school of computer science.

Finally, Chapter 5 highlights the conclusions along with some future work.

Appendices of the source codes of the CGI scripts are given at the end of the document.
Chapter 2 Fill-Out Forms and the Common Gateway Interface

Most of the traditional publishing media communicate in only one direction. People can only read them; they can not interact with them because these media are designed to carry information from the provider to consumers. Most of the information documents on the Web are published in this static way. However, the distributed design of the Internet has the potential to break that model and allow people to develop new patterns of communication. The recent emergence of embedded HTML Fill-out Forms and Common Gateway Interface has realized some of this potential. The proper use of forms and CGI script can make Web pages dynamic.

This chapter introduces the concept of HTML Fill-out Forms and CGI by focusing on the implementation aspects.

2.1 Fill-out Forms

Fill-out Forms are used in HTML documents to allow the user to enter information directly on a Web page, and then have it transmitted to a server. This server will process the information and forward it to the page owner or to some other mechanism.

Forms can contain a number of elements, some essential, and others optional, depending on the purpose of the form. Any HTML page can be a form, or have a form as an element within the page. The following summary of HTML Fill-out Forms is based on HTML 2.0 specifications [3].

The Basic Structure of a Form

Figure 4 is an example of a log-on form on the Web to restrict access to certain Web pages:
UNIVERSITY OF WINDSOR

SCHOOL OF COMPUTER

Log-on form

Please enter your **user name** and **password**, then press log-on.

User Name:

[Blank]

Password:

[Blank]

Log-on

To reset the log-on form, press this button: **reset**
Appendix A is the source code for that form.

Even though HTML forms have gained a reputation for complexity, in reality the user only needs to learn a few simple elements to master putting them together effectively. Every form has the same basic structure. A form is denoted by the element `<form>`. The `<form>` element has two qualifiers, action and method. Action is the URL of the program that will process the completed form. Forms are generally processed by programs written with CGI, the Common Gateway Interface.

The method is the way in which the information will be moved from the client to the server program that will process it. The options are GET or POST. The GET method tells the client to embed the data from the form inside the URL. If a form uses the POST method, the server sends the submitted data as if it were typed in by the user. In this example and also in the thesis work, POST method is used since it is strongly recommended by the designers and is already used in many existing applications.

**Inputting the Information**

Once the form is defined, input fields can be added. These fields allow the user to enter information which is then assigned to a specified variable name. This is done using the `<INPUT>` element which has the following options:

- **TYPE**: Several types of data can be used for the variable. They are shown as follows:
  - `TYPE="text"` is characters entered on the screen which will be displayed as the user enters them.
  - `TYPE="password"` is also a character string. However, it will not be displayed.
  - `TYPE="checkbox"` is a box that can be selected. It translates to a yes or no when the information is formatted, depending on whether the box was selected or not.
- **TYPE=“radio”** allows the user to select one of several radio fields. A radio field is associated with a small button. Only one button can be selected at a time.

- **TYPE=“reset”** will reset the entire form to its default values.

- **TYPE=“submit”** is a button that causes the form to be submitted to the server for processing. Because of this feature, it serves an important purpose in almost every form.

- **NAME** is a string of text that identifies the input associated with a specific type. This is for identification purposes during processing.

- **SIZE** determines the maximum size allowed for text or password input.

The tag `<TEXTAREA>` allows the user to enter text freely within a certain area of the screen. The user can define the name, rows and columns of this tag.

**Building a Form**

An HTML form can reside anywhere within an HTML document as an element, or can even be created by a CGI script. A document can have several forms within it, and each can perform different tasks or ask for different information.

**2.2 Common Gateway Interface**

The **Common Gateway Interface** or **CGI** is a standard for external gateway programs to interface with information servers [1]. Gateways are programs or scripts which handle information requests and return the appropriate document or even generate a document on the fly.

**Programming With CGI**

Any programming language that can access UNIX environment variables can use the CGI. CGI programs can be written in C, C++, perl, TCL, or Bourne and C shell
scripts. All of these languages, except C and C++, are interpreted. The advantage of using perl or TCL languages is that many CGI programming tasks require a lot of string handling, and these languages have powerful built-in functions for manipulating text. In addition, most interpreted languages let you write code for the interpreter on the fly, which is extremely useful for many gateways. However, the dynamic interpretation features of these languages can be a security hazard, and, also, programs written in script languages often run slower than those written in a compiled language. That’s the reason why C is chosen in the thesis work.

Most servers have restrictions on who can create CGI programs and from which directories they can be run. Allowing clients to run arbitrary programs on the server is quite dangerous from a security standpoint, and server administrators don’t give this permission to every user. Most servers have a single valid directory for CGI scripts, which is represented by the URL /cgi-bin/. Usually only the server administrator and authorized users can write to this directory.

There are two important rules to follow when writing CGI programs:

- Any data or text is written for the client to the standard output.
- Any output must be preceded with a Content-type line, defining its MIME type, followed by a blank line.

Following these two rules, new Web pages can be generated with CGI scripts. The user simply writes any data he wants the client to receive to the standard output, preceded by its MIME type, and the server does the rest.

**Handling HTML Forms**

HTML forms create interactive areas in the Web pages. The data entered into a form by a user is assembled by the browser into a series of name/value pairs. Then

16
these name/values pairs are sent to the server to be processed by a CGI script which is specified by the ACTION attribute in the form. The CGI program can then act upon the input, and perform a variety of tasks, such as sending mail messages, querying or updating some files, or any task performed by a CGI script on the server itself.

For the POST method, the server sends the data that it received from the client to the handling CGI script on its standard input. It also sets the REQUESTED_METHOD environment variable to POST. But the HTTP server sends no signal that it has finished transmitting data. Instead, the HTTP server sets the CONTENT_LENGTH environment variable, which gives the amount of input the script should read in bytes. When the script has read that amount of data from the standard input, it should stop reading and begin to process the actual data from the form. Processing the data requires that the script decode the URL encoding which includes four steps:

1. Break up the pairs of HTML form variables and their values, which are separated by ampersands.
2. Separate each HTML form variable from its value; the variables and their values are linked by equal signs.
3. Convert all hex sequences in each of the variables and the values to their ASCII equivalents.
4. Replace all plus signs in each of the variables and values with spaces.

Since every CGI script will counter such data processing, we put all the functions together as Appendix B shows. After these steps, the CGI script programmers can do whatever they like with them. The output of a CGI script also follows two rules: precede data with a Content-type header, specifying the MIME type, and a blank line. Appendix C is the source code of CGI script written in C which handles the log-on form.
Chapter 3  An Environment for the Development and Modification of Dynamic Hypertext Information Structures

With CGI and HTML Fill-out Forms, new HTML pages can be generated on the fly, which is an extremely useful and powerful capability. With dynamic documents, people can alter the appearance, interface and features of their Web pages as they desire. Based on these concepts, an environment for the development and modification of dynamic hypertext information structures was designed and developed. It is an interactive environment residing on the Web. The main objectives of creating such an environment are:

- To support a group of people communicating through the Web.
- To build dynamic hypertext information structures.
- To present the current state of changing information resources.
- To save people the trouble of changing information on the Web and creating pages that would be difficult, annoying or impossible to write by hand.

So the environment is a software package that provides documents with read/write access on the World Wide Web. A document in this environment consists of pages with text and hypertext links. Pages can be created, changed, and added to interactively, using a standard WWW client. All changes and additions are visible immediately to all users. Thus this environment is not just a place for storing information; it is also a medium for direct communication among its users.

This chapter covers the design of such an environment on the Web and explains how it is accomplished.
3.1 System Design

During the design phase of the environment, it was decided that the environment should have the following features:

User Interface

- Access to all functions should be possible using standard WWW clients
- Easy to use point-and-click interface for editing
- No knowledge of HTML required when editing

Document Structure

- A document in the environment consists of pages
- The system operator maintains the document home page
- The user group maintains all other pages of the document
- Users can add and delete pages
- Users can add, edit, and delete text on pages
- A page contains:
  - A Page title
  - A toolbar for accessing various functions
  - Zero or more items
- An item can be
  - A piece of text
  - A link to another page
  - A link to a place elsewhere on the Web
Navigation Tools

- [Home], [Top] and [Up] buttons on each page for moving through the page tree
- Index tree of the complete page structure

Basic Editing functions

- Add text anywhere on any page
- Edit text written earlier
- Create link to a new page
- Delete item on a page

Text Typesetting Options

- Normal/bold/italics
- HTML code
- With or without author’s name

Ways of Accessing the environment and the files in the environment

- Log-on using a log-on form
- Files can only be read through this environment
- All authorized users are responsible for maintaining the information structure in the environment

Miscellaneous

- Re-load the current page to show any changes
- On-line user’s guide
- Support a multiple-readers/one-writer file access policy
3.2 System Development

Except for the first log-on page of the environment, all pages are generated by CGI scripts on the fly. In this way, all changes to the information structure are immediately visible to all users who are accessing the system.

The Log-on Page

For the purpose of security, only authorized users can gain access to the information in the environment. This is accomplished by a log-on form as the following figure shows.

Appendix A is the HTML source code for this log-on form. As we mentioned before, this is the only static HTML page in the whole system. Appendix C is the CGI script which checks the validity of the password entered by the user. If the password is correct, the script generates the next page; otherwise it returns an error message to the user. See Appendix F for the source code.

Pages of the Dynamic Hypertext Information Structure Builder

The dynamic hypertext information structure is made of pages. All pages maintained by the system can have two types of elements on them: the toolbar and information items as shown in the Figure 5. Every page has its own page title. Below the page title is the toolbar which has two rows of buttons (WWW links). Clicking on each button will invoke the corresponding program to execute the request and go to the appropriate page.
Counselling

[Index] [Help] [Reload] [Back]

* Addtext * Newpage * NewLink *

-- Name List --

Lisa Lee, Joe Wang, Lily Smith. (Written by Dr. Smith / Mon Oct 23 06:39:48 1995)

-- Hyper-Link --

Registration Office (Created by Dr. Smith / Mon Oct 23 06:40:33 1995)

Appointment
A page can contain several information items below the toolbar. An item is either a link to another page or some information written by a user. This information usually takes the form of a text item, containing the name of the author and the date it was written. At the end of every item (except links to pages) are three buttons: the pencil button, the capital character A button, and the scissors button. The item of a link to another page doesn’t contain the first two buttons. With the pencil button, the user can re-edit a text item. With the ‘A’ button the user can append his response to the original text. With the scissors button, the user can delete unwanted items from the page.

At the bottom of the page are the navigation buttons of the system. Clicking the first button, the user can always get back to the ‘home’ page of the dynamic structure. The second one points to the top of the hyper-document created by the users of the system. The third one will go to the parent page of the current page.

The homepage and the top page of the hyper-document system are created by the system. The information structure beneath it is made by user community.

Creating a Page  On the toolbar on each page, there is a button called Newpage. By clicking this button, the system will present a form for creating a new page as the Figure 6 shows. The CGI script for this action is in Appendix D.
Creating a New Page

Operation:

- Create a link to a new empty page

Put the page title below:

Or use the back function to go to the previous page

Create the link  reset

Figure 6 Form for creating a new page

Appendix E is the source code of the action program to take the input data from the form and create a new page which is presented to the user after the user clicks the Create the link button.
Once the page is created, the user cannot change it. At the end of the page link is the scissor button which is used to delete the page. However, the user can delete the page only when the page is empty. This is the way the system protects files from accidental operations. Appendix G is the CGI script source code to delete a empty page.

**Opening a Page** The system simultaneously generates a new page for the user and presents it to the user. At the same time, it creates a hypertext link on the parent page pointing to the new page. All pages in the system are linked in this way. When the hyperlink is clicked, a CGI script is invoked to present the desired page to the user. See Appendix F for the source code of this action. When a page is opened, the toolbar, all items on the page, and the navigation buttons are also presented to the user.

**Creating a Hyperlink to Anywhere On the Web**

A page in our environment can contain an HTML file or a plain text file, links to its children pages, and hypertext links to anywhere in the Web. The system has a special facility to help the user create hyper-link to other locations on the Web without the knowledge of HTML.

When the Newlink button is clicked, a form for creating Hyperlink is presented as the following figure shows.
Appendix H is the CGI script code for creating the form. Appendix I is the source.
code to create the link and put the link on its parent page. A link is followed by three buttons: the pencil button for changing it, the capital character A for appending notes to the link, and the scissors button to delete that link.

### 3.3 Text File Manipulation in the System

Text manipulation is one of the focuses during the system design and development. Currently, all users are responsible for maintaining the information in the environment. Issues about multiple readers and writers are discussed here.

**Putting a Text on the Web through our Environment**

The traditional way of putting information on the Web is for the user to open an editor to edit the HTML or plain text file. The user can use the open file functionality on the browser to see his or her file. If the user wants others to read this document, he or she has to link the document to be linked somewhere on a Web server. If using our environment, the procedure is quite simple. The user can put messages on the Web directly using a browser.

By clicking on the Addtext button, a CGI script (see Appendix J) is invoked to generate a form for the text adding page as the Figure 8 shows.
Text adding page

Select one text style: ◊ Normal ◊ Italic ◊ Bold ◊ HTML code

Put the title into the following area:

[Input Field]

Put your name into the following area:

[Input Field]

Put the content into the following form:

[Text Area]

Add text [Clear form]

Document: Done.

Figure 8 Form for putting new text on the Web
In the form, the user can choose his or her favorite text style among normal, italic, bold or HTML code. Then the user simply follows the instructions on the form to put appropriate text contents into the area. The author's name is optional. When the user finishes typing, he or she clicks on the Add text button to invoke a CGI script (see Appendix K) to save the file and put the file to the Web. All other system users can see the text immediately.

Each text item is followed by three buttons: the pencil button for re-editing, the capital character A button for appending a reader's response to the original text, and the scissors button to delete the text item.

The text is readable and appendable by multiple users. When more than one user try to append messages to the same text file at the same time, all the appending messages are saved but the order is slightly different. Figure 9 is the form to append a response to an example text. Appendix L is the CGI script to generate such a form.
Appending text page

-- Name List --

Lisa Lee, Joe Wang, Lily Smith. (Written by Dr. Smith / Mon Oct 23 06:39:48 1995)

Use the form below to add new text.

Text style: Normal ❖ Italic ❖ Bold

Put your name into the following area:

Append text  Clear form

Figure 9 Form for appending a response to the text item
Locking Facilities for Multiple Readers and Single Writer

Each user has access to all information items in the environment. The system also allows the user to edit the text written earlier by another author. Through this, the system supplies co-authoring facility to the user community. However, when more than one user is trying to edit the same text and save their own changes concurrently, some changes may be lost. This is because the UNIX system allows more than one program to open a file for writing at the same time. This means that if two programs write data to a file at the same time, the data from one write will overwrite the data from the other. The system simply doesn’t check to see if a file is already opened when someone else opens it for writing or truncates it.

In order to avoid this, a multiple-readers/single-writer lock is necessary for editing text. Since the editing functionality is achieved through two CGI scripts — one for creating an editing form, and another one for writing the changes from the form to the text — as soon as one user enters the editing form for the text, he or she should put an exclusive lock on that text which prevents other users from editing the same text. After he or she finishes the editing, he or she presses the submit button to invoke the action CGI script to save the changes to the file. Once the saving CGI script is done, the lock file will be released. Then other users can have writing access to that text which has been edited.

Unfortunately, the UNIX system we are using doesn’t have such facility. So we have created our own locking mechanism. Once the user enters the editing form, the system creates a lock file for the text in a special directory /lock/. If other users want to edit the text, the system checks the /lock/ directory to see if there is a lock file for that text. If there is, they will get an error message stating that they cannot enter the editing form right now. Thereby, only one user can gain the access to the editing form for that text. After the user finishes editing, another CGI script is invoked to save the
change. Only then, does the system remove the lock file from the lock directory. Now other users can edit that text again.

Appendix N is the CGI script to generate the text editing form and the file writing lock. Appendix O is the source code to save the changes and release the lock. When the user re-edits a file, he or she can only change the original contents, which means the appending part from other users remains the same.

This locking mechanism also works for the deleting action. If one user is editing the text, no other users can delete this text. Appendix P and Q are the source CGI code for deleting text from a page.

3.4 Implementation Problem: Browser Caching

During the development of the system, a browser caching problem was identified. Most browsers (including the one we are using, “Netscape”) have a local cache mechanism that automatically stores the last few pages. This allows the user to backtrack during a session without server access. As long as the server only supplies static, frozen information resources, this causes no problems at all. However, in our environment, the state of the information is dynamic which means the information keeps changing from time to time. This state-information may become inconsistent with that in the front-end especially when data modification happens.

This problem exists because Client/Server protocols like Hypertext Transfer Protocol of the World-Wide Web are stateless. The reaction of the server depends on the request only. To access the server, client programs (browsers) are required. Such browsers are stateless.

So far, even though different approaches have been suggested, the problem of entering or updating data using stateless browsers has not yet been solved[6].
One attractive way is to re-design the back function supplied by the Netscape. Unfortunately, this is impossible. We cannot disable the go back button because the system allows the user to create outside environment links to any locations on the Web, and the user should be able to use the back function to go back to our environment pages. The approach we adopted in this thesis is to design a new navigation system. As shown in the following figure, each page has three initiations buttons at the bottom. Besides these, there are three buttons on top of the page: index, reload and back.

\[ \text{Netscape: Child page} \]

**Research**

[Index] [Help] [Reload] [Back]

*Addtext* *Newpage* *NewLink* *

Figure 10 Example page
Among the three buttons on the bottom of the page, the first one goes to the home page of the environment, the second goes to the top page of the dynamic information structure, and the third one goes to the parent page of the current page. These buttons actually execute CGI commands to open the desired page and download the most recent information items on that page. In the toolbar on top of the page, the back button is exactly the same as the third button on the bottom.

![Netscape: Index Page](image)

**Directory Index**

/Top Page

/Faculty Members Administrative Tasks

/Research

/Teaching

/Course Outline

/Schedule

/Course-60-100

Figure 11 Dynamic page directory tree
Clicking the index button presents a dynamic page tree directory to the user. It helps the user keep track of the changes to the page structure in the environment. Appendix R is the source code for generating such a dynamic tree. Figure 11 shows part of an index directory tree for an information structure.

For the local browser caching problem, we designed a reload button for each page generated. The button simply reloads the current page so the user can see the changes made by other users. With the reload button, we allow the user to use the Netscape back function to go back to our environment, provided he presses the reload button as soon as he comes back.
In this chapter, we give an example of applications of our environment.

4.1 Home page

![Netscape Welcome page](image)

Welcome to the Dynamic Hypertext Information Structure Creating Environment

*Index* *User's Guide* *Dynamic IS* *Bulletin Board*

Figure 12 Home page

After the user successfully logs on to the system, the home page (see Figure 12) is presented. Four buttons are available on the toolbar on the home page. By clicking on
these buttons, the user can check the current page-index tree, read the user’s guide, go
to the bulletin board or enter a dynamic information structure. We will now look at the
various pages of the bulletin board and environment by focusing on actual example usage.

4.2 Bulletin Board

By clicking on the Bulletin Board button on the home page, the user enters the
bulletin board page. The bulletin board is designed for the purpose of posting messages
to a group of people and collecting responses. Figure shows what a bulletin board looks
like.
Under the title are three buttons: Home, Help and Put Message. The first button will navigate the user to go back to the home page of the environment. When the user clicks on the Put Message button, a form for putting messages into the bulletin board is presented to the user as Figure shows. The user can choose his or her favorite text style and follow the instructions to put the text into the form. He or she can also use the Clear form button to clear the whole form. After the user finishes typing text into the form, he or she presses the Put to Bulletin Board button. Now the bulletin board page appears again. and this time, it includes the message title the user just enters in, the author's name and date as Figure 13 shows.

Suppose we want to read this message, we can just click on the title and a page with the full contents of the message appears as Figure 14 shows.

Below the message, there are two buttons: capital character A button and scissors button. The first one is used to collect responses from readers. By clicking on it, a page for appending responses presents as Figure shows. The reader can put their response and name which is optional to the form, and then press the Append message button. Now we go back to the bulletin board again. This time, when the user clicks on the message title, he gets not only the original message, but also the response from other users.

The scissors button is used to delete messages from the bulletin board. When the user clicks on it, a text deleting page is presented with the original message and response on it. On this page the user confirms that he really wants to delete that message by choosing the Delete this item operation and then pressing the delete this item button. If the user changes his mind, he can always use the back function supplied by the Netscape to go back to the previous page.
Multiple users can access the bulletin board at the same time. If more than two users are appending responses to the same message, all the responses are saved.

4.3 Dynamic Information Structure

By clicking on the review button on the home page, the user can go to the top page of the dynamic hypertext information structure environment as the following figure shows. Below the page title, there are two rows of buttons. On the first row, there is a reload button which will re-load the current page so the user can get the most recent changes
on that page. This is especially useful when the user is reading the page, and other users make some changes on the same page.

![Netscape: First page](image)

**Top Page**

[Index] [Help] [Reload]

*Addtext* *Newpage* *NewLink* *

![House icon](image)

Figure 15  Top page of the environment

The second row has three buttons: Addtext, Newpage and Newlink.

The Addtext button is used to add text on the current page. The Newpage button is used to create a new page under the current page. The Newlink button can be used to create hyper-link to anywhere on the Web.

**Examples of Application Development**

Suppose we wish to build a dynamic hypertext information structure for faculty members administrative tasks of the School of Computer Science on the Web using the
environment. First we analyze the administrative tasks of faculties as the following figure shows:

![Diagram of Faculty Members Administrative Tasks]

Figure 16 Faculty members administrative tasks

We are going to build a structured page tree which resembles the activity tree shown in the above figure.

**Building A Dynamic Hypertext Information Structure for Faculty Members of the School of Computer Science**

For each activity, we create a page. From the top page, we click on the Newpage button. As the following figure shows, a form for creating a new page is presented.
Creating a New Page

Operation:
- Create a link to a new empty page

Put the page title below:

Or use the back function to go to the previous page

[Create the link] [reset]

Figure 17 Form for creating new page

We put "Faculty Members Administrative Tasks" into the page title area. Then click the Create the link button. A new page with the title of "Faculty Members Administrative Tasks" is presented as the following figure shows.
Faculty Members Administrative Tasks

[Index] [Help] [Reload] [Back]

* Addtext * Newpage * NewLink *

Figure 18 A page

This page is a slightly different from the top page. At the bottom of the page, there are three navigation buttons instead of one. When clicking on the first one, the user will go right back to the home page, on the second one, to the top page, and on the third one, back to the previous page. In this case, the user will go back to the top page. The rest of the pages will be built up in the same way.

Finally, the Faculty Members Administrative Tasks tree is linked as the following figure shows:
Faculty Members Administrative Tasks

[Index] [Help] [Reload] [Back]

* Addtext * Newpage * NewLink *

Research

Teaching

Grad

Undergrad

High School Liaison

Figure 19 Faculty members administrative tasks page
Each page title is a hypertext link to the page. At the end of each page item is a scissors button which allows the user to delete the corresponding page. However, when a page is not empty, the system will not allow the user to remove that page. Besides links to children pages, each page may also have hyper-link to anywhere in the Web, and text messages on them.

On any page, there is a button called "Index". When the user clicks on it, a page with a dynamic page tree index is presented. Here, dynamic means that if a user changes the page structure, this index tree will reflect it. Figure 11 shows part of the dynamic page tree index of the dynamic information structure for faculty members administrative tasks.

Figure 20 is the Counselling page with a student name list and a hyper-link to the registration office.
Counselling

[Index] [Help] [Reload] [Back]

*Addtext* *Newpage* *NewLink*

--- Name List ---

Lisa Lee, Joe Wang, Lily Smith. *(Written by Dr. Smith / Mon Oct 23 06:39:48 1995)*

--- Hyper-Link ---

Registration Office *(Created by Dr. Smith / Mon Oct 23 06:40:33 1995)*

Figure 20 A page with text and hyper-link
Chapter 5  Conclusions and Future work

This chapter explains the conclusions formed as a result of the thesis work and outlines some important extensions.

5.1 Conclusions

In the thesis work, we created an environment for the development and modification of dynamic hypertext information structures on the Web. This environment is based on a Client/Server model using Hypertext Transfer Protocol (HTTP). The information structures can be shared by a group of people. Every member of the group can manipulate all the information in the structure. We focused on creating an interactive environment between the user groups and the Web server. Like all Web documents, this environment consists of pages with text, graphics, and hypertext links to other pages. Unlike most Web documents, a document in this environment does not consist of read-only, frozen information. This environment maintains read/write documents that can be changed, added, and deleted interactively, using a standard Web client. All authorized users can create structured pages, and write, edit, and delete text on pages without knowing Hypertext Markup Language and Hypertext Transfer Protocol. Special system facilities help the users keep track of the changes and additions to the information structure.

We emphasized "Dynamic" in the design phase. Except for the first log-on page to the environment, all the other pages are generated on the fly with CGI scripts, which means when a user downloads a new page, he or she is actually invoking the server to execute the CGI script to check that page directory and generate an HTML page in the format the client can understand. In this way, all information items can be changed by each member in an authorized user group, and the changes are visible to all other
members immediately. When the user uses our navigation facilities to go to another page, he will see the most recent changes on that page. Because of the local caching mechanism of most browsers, when the user is reading the current page and another user changes the same page, the reading side won’t feel the changes. We designed our reload functionality to let the user re-download that page when he feels necessary. The reload button will also help the user to reload the current page when he uses the Netscape built-in back function to go back to the environment from an outside link. In order not to cause any confusion and loss, when a user re-edits an item on the page, other users can still read and append responses to that item; but before that user finishes his or her editing, others are forbidden to edit and delete the same item.

The work has shown that it is possible to build a WWW environment which

1. Allows people to share the creation of information.
2. Improves the interactive communication between the Web and the Web users with an easy to use point-and-click interface.
3. Allows people to construct structured Web pages without the knowledge of HTML.
4. Allows the World-Wide Web to be used as a collaborative tool.

5.2 Future Work

Currently, every one in an authorized user group has access to all pages in the environment. Each of them can add, change and delete text on all pages and can also create new pages, or delete pages. This is not convenient if it is appropriate to restrict access to certain information items and certain pages in the environment. Future work could include the design of system facilities to help users set up different access restrictions for different pages and information items on each page.
Another area of future work would be to build facilities to move pages and items. Currently, pages and items on the pages cannot be moved within the information structure. Also, the capability of copying a set of structures and pasting them to another page should be considered.
HTML Code of the Log-on Form

<TITLE> Log-on form </TITLE>
<H1><IMG SRC="http://www.cs.uwindsor.ca/units/cs/logo.gif" ALT="[SCHOOL OF COMPUTER SCIENCE]"> Log-on form</H1>

Please enter your <B>user name</B> and <B>password</B>, then press <TT>log-on</TT>.

Please enter your <B>user name</B> and <B>password</B>, then press <TT>log-on</TT>.

"POST" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/logon">

<DT><B>User Name:</B> </DT>
<DD><INPUT SIZE="20" NAME="uname"> </DD>

<DT><B>Password: </B> </DT>
<DD><INPUT TYPE="password" SIZE="20" NAME="passwd">
<INPUT TYPE="submit" VALUE="Log-on">
<DT>To reset the log-on form, press this button: <INPUT TYPE="reset" VALUE="reset">

</DL>
</FORM>
Function Library for CGI Scripts

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>

char *path;

void plustospace(char *str) {
    register int x;
    for(x=0;str[x];x++) if(str[x] == ' ') str[x] = ' ';
}

/*open each file on each page */

int prfile (char *fname, struct stat *staptr, int flag){
    FILE *fp;
    char ch, *p, temp_path[500], *file_path, *title;
    int cl;

    /* get the path name of the current fname
     (can be the current directory name or the file under the directory)*/
    strcpy(temp_path, fname);
    file_path = temp_path;
    if (strcmp (path, file_path)) {
        cl = strlen(fname);
        p = file_path;
        p = p + cl - 1;
        while (*p != '/') p--;
        *p = NULL;
    }

    /* open the file in the current directory */
    if (strcmp (path, file_path) == 0) {
        if (flag == FTW_F) {
            fp = fopen(fname, "r");
            ch = getc(fp);
            while (!feof(fp)) {
                printf("%c", ch);
                ch = getc(fp);
            }
            fclose(fp);
        }
    }

    printf("<A SRC=" http://www.cs.u windsor.ca/cgi-bin/wangd/Info/edit_text_form.html"", fname);
    printf(" <IMG SRC=" file://home/cs/misc/www/grads/wangd/icon/pen.gif"">");
    printf("</A>");

    printf("<A SRC=" http://www.cs.u windsor.ca/cgi-bin/wangd/Info/append_text_form.html"", fname);
    printf(" <IMG SRC=" file://home/cs/misc/www/grads/wangd/icon/a.gif"">");
    printf("</A>");

}
printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/delete_text_form%es">", fname);
printf(" <img src="/file:/home/cs/misc/www/grads/wangd/icon/cissorow.gif">");
printf("</A>");}
else if ( (flag ==FTW_D) && (fname != path)) {
p=fname;
cl=strlen(fname);
p=p+cl-1;
while (*p=='/') p--;
p++;
sprintf(temp_path, p);
title=temp_path;
printf("<P>");
printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/open_page">");
printf("%s", fname);
printf("\n"); plustospace(title);
printf("%s", title);
printf("</A>");
printf("</P>");

printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/delete_dir%es">", fname);
printf(" <img src="/file:/home/cs/misc/www/grads/wangd/icon/cissorow.gif">");
printf("</A>");
}
return(0);
}

char *fmakeword(FILE *f, char stop, int *cl) {
    int ssize;
    char *word;
    int ll;

    ssize = 102400;
    ll=0;
    word = (char *) malloc(sizeof(char) * (ssize + 1));

    while(ll) {
        word[ll] = (char) fgetc(f);
        if(ll==ssize) {
            word[ll+1] = '\0';
            ssize=102400;
            word = (char *) realloc(word,sizeof(char) *(ssize+1));
        }
        -*cl;
        if((word[ll] == stop) || (feof(f)) || (!(*cl))) {
            if(word[ll] != stop) ll++;
            word[ll] = '\0';
            return word;
        }
        ++ll;
    }
}

char x2c(char *what) {
    register char digit;
digit = (what[0] >= 'A' ? ((what[0] & 0xdf) - 'A')+10 :
(what[0] - '0'));
digit *= 16;
(what[1] - '0'));
return(digit);
}

void unescape_url(char *url) {
    register int x, y;

    for(x=0, y=0;url[y]++x, ++y) {
        if((url[x] = url[y]) == '&') {
            url[x] = x2c(&url[y+1]);
y+=2;
        }
    }
    url[x] = '\0';
}

char *makeword(char *line, char stop) {
    int x = 0, y;
    char *word = (char *) malloc(sizeof(char) * (strlen(line) + 1));

    for(x=0;((line[x] && (line[x] != stop));x++)
    word[x] = line[x];

    word[x] = '\0';
    if(line[x] ++x;
y=0;

    while(line[y++] = line[x++]);
    return word;
}

/* print out the header for each page */
void header(char *path_info){
    char *p, tmp_path[500], *back_path;
    int cl;

    strcpy(tmp_path, path_info);
    back_path=tmp_path;
    cl=strlen(path_info);
p=back_path;
p=p+cl-1;
    while (*p=='/') p--;
    *p=NULL;
p++;

    printf("<DL compact> <DD>\n\n    printf(" [<A href="http://www.cs.uwindsor.ca/cgi-bin/
wangd/Info/index">Index</A>]\n\n    printf(" [<A href="http://www.cs.uwindsor.ca">Help</A>]\n\n    printf(" [<A href="http://www.cs.uwindsor.ca/cgi-bin/
wangd/Info/open_page">Reload</A>", path_info);
    if ((p="/Homepage" && (cl=55))
    printf(" [<A href="http://www.cs.uwindsor.ca/cgi-bin/
wangd/Info/open_page">Back</A>", back_path);
    printf("</DL>\n\n    printf("<B>\n\n    printf("**\n\n53
printf("<A href="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/add_text_form%s">Addtext</A>", path_info);
printf(" *");
printf(" <A href="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/create_page_form%s">Newpage</A>", path_info);
printf(" *");
printf(" <A href="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/create_link_form%s">NewLink</A>", path_info);
printf(" *");
printf("<B>");
}

/* print out the tail for each page */
void tailer(char *path_info){
char *p;
int cl;

cl=strlen(path_info);
p=path_info;
p=p+cl-1;
while (*p=='/') p--;
*p=NULL;
p++;

if (p!="Top+Page" && cl!=55) {
    printf("<P>");
    printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/home_page">"");
    printf(" <img src="file:home/cs/misc/www/grads/wangd/icon/go_home_btn.gif"">");
    printf("</A>"");
    printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/review">"");
    printf(" <img src="file:home/cs/misc/www/grads/wangd/icon/go_top_btn.gif"">");
    printf("</A>" );

    printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/open_page%s">", path_info);
    printf(" <img src="file:home/cs/misc/www/grads/wangd/icon/go_up_btn.gif"">");
    printf("</A>");
} else {
    printf("<P>");
    printf("<A HREF="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/home_page">"");
    printf(" <img src="file:home/cs/misc/www/grads/wangd/icon/go_home_btn.gif"">");
    printf("</A>" );
}
# CGI Script for Log-on Checking

/* This is the log-on file which checks the legal password for users */

#include <stdio.h>
#include <stdlib.h>

#define PASSWORD "aub"
#define MAX_ENTRIES 10

typedef struct {
    char *name;
    char *val;
} entry;

car *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char *x2c(char *what);
void unescape_url(char *url);
void plusospace(char *str);

main(int argc, char *argv[]){
    entry entries[MAX_ENTRIES];
    register int i, m=0;
    int cl;

    printf("Content-type: text/html;charset", 10, 10);
    cl=atoi(getenv("CONTENT_LENGTH"));

    /* sort the stdin file into struct */
    for (x=0; cl & x < feof(stdin); x++) {
        entries[x].val = fmakeword(stdin, ",", &cl);
        plusospace(entries[x].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val, ";");
    }

    if (strcmp(entries[1].val, PASSWORD)) {
        printf("<h1>Logon error</h1><p>");
        printf("Unknown user name or wrong password. <p>");
        printf("Use the 'back' function of your client to try again.<p>");
        exit (1);
    }

    printf("<img src="http://www.cs.uwindsor.ca/units/cs/logo.gif"
            alt="[SCHOOL OF COMPUTER SCIENCE]">");
    printf("<title>Welcome page</title>");
    printf("<h1>Welcome to the Dynamic Hypertext Information Structure Creating Environment</h1><p>");
    /*print("<dl compact><dd>");
    printf("<a href="http://www.cs.uwindsor.ca">Index</a>\n"");
    printf("<a href="http://www.cs.uwindsor.ca">Help</a>\n"");
    printf("<a href="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/Review">Review</a>\n"");
    printf("<a href="http://www.cs.uwindsor.ca/cgi-bin/wangd/BB/bulletin_board">bulletin board</a>\n"");
    printf("</dl>\n"*/
    printf("<b>*\n"");
    printf("<a href="http://www.cs.uwindsor.ca/cgi-bin/
            wangd/Info/index">Index</a>\n"");
    printf("<a href="http://www.cs.uwindsor.ca">
            User's Guide</a>\n"");
}

55
printf("<A href="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/review">Review </A>" );
printf("<A href="http://www.cs.uwindsor.ca/cgi-bin/wangd/BB/bulletin_board">Bulletin Board </A>" );
printf("</B><P>" );

;/* int rind(char *s, char c) {
    register int x;
    for(x=strlen(s) - 1; x != -1; x--) {
        if(s[x] == c) return x;
    }
    return -1;
}*/
Form for Creating New Page

/* This program generates a form for creating a new page. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <ftw.h>
#include <time.h>

main(int argc, char *argv[]) {
    char *path;

    printf("Content-type: text/html\r\n", 10, 10);
    path=getenv("PATH_INFO");
    printf("<TITLE>Creating new Page</TITLE>\r\n");
    printf("<H1> Creating a New Page</H1>\r\n");
    printf("<HR>\r\n");
    printf("<FORM METHOD="POST" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/create_page\r\n%">\r\n", path);
    printf("<DL>\r\n");
    printf("<DT><B>Operation:<B></B>\r\n");
    printf("<DD><INPUT TYPE="radio" NAME="action"\r\n");
    printf("Create a <B>link</B> to a <B>new</B> empty page\r\n");
    printf("<P>\r\n");
    printf("Put the page title below:<P>\r\n");
    printf("<DD><INPUT SIZE="50" NAME="text"\r\n");
    printf("Or use the <B>back</B> function to go to the previous page<P>\r\n");
    printf("<DL>\r\n");
    printf("<HR>\r\n");
    printf("<INPUT TYPE="submit" VALUE="Create the link"\r\n");
    printf("<INPUT TYPE="reset" VALUE="reset"\r\n");
    printf("</FORM>\r\n");
}

/* This program creates a new page for the information structure. */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <ftw.h>
#include <time.h>
#include <string.h>

#define MAX_ENTRIES 100

typedef struct {
    char *name;
    char *val;
} entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plusospase(char *str);
int profile(char *fname, struct stat *statptr, int flag);
void header(char *path_info);
void tailer(char *path_info);

char *path;

main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x=0;
    int cl, random;
    char ch, buf[500], buf1[500], buf2[500], *p, *slash, *d;
    char name_buf1[500], name_buf2[500], name_buf3[500], *pf;
    int m, n;
    char *constant;
    char *pagename, page_title[500], *tmp_page_title, *pt, *new_page_path;
    FILE *fd;
    time_t time_now;
    srand((int) time(NULL));

    printf("Content-type: text/html;charset",10,10);

    cl = atoi(getenv("CONTENT_LENGTH"));

    path=getenv("PATH_INFO");
    new_page_path=getenv("PATH_INFO");
    pt=fmakeword(stdin,'&',&cl);

    /* plusospase(pt);*/
    unescape_url(pt);

    strcpy(name_buf3, pt);
    pf=name_buf3;
    while (*pf != '=' pf++;
    pf++;
    plusospase(pf);
strcpy(name_buf1, pt);
c1=strlen(pt);
m=0;
n=0;
while (n <= c1) {
    if (name_buf1[n] != '\') {
        name_buf2[m]=name_buf1[n];
        m++; n++;
    } else if (name_buf1[n] == '\') {
        name_buf2[m]='\';
        m++;
    }
    name_buf2[m]=name_buf1[n];
    m++; n++;
}
pagename=name_buf2;
    while (*pagename != '=') pagename++;
pagename++;

    while (*pt != '=') pt++;
    pt++;
    /*change to the current page directory */
    chdir(path);

    /* create a directory for the new page */
    sprintf(buf, "mkdir %s", pagename);
    system(buf);

    /* change mode of the new directory */
    sprintf(buf, "chmod a+rw %s", pagename);
    system(buf);

    /* get the path for the new page*/
    slash="/",
    strcat(new_page_path, slash);
    strcat(new_page_path, pagename);

    /* get the title for that page */
    cl=strlen(path);
    p=path;
    p=p+cl-1;
    while (*p!='/') p--;
    p++;
    tmp_page_title=p;
    strcpy(page_title, tmp_page_title);
    plustospace(page_title); /*
    plustospace(pagename);
    printf("<TITLE>Child page</TITLE>");
    printf("<H1>%s</H1>", pf);
    header(new_page_path);
    ftw(new_page_path, pf, 30);
    tailer(new_page_path);"
CGI Script for Opening A Page

/* This program is used to open a page*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <ftw.h>
#include <time.h>

int profile (char *fname, struct stat *statptr, int flag);
void plustospace(char *str);
void header(char *path_info);
void tailer(char *path_info);

char *path;

main(int argc, char *argv[]) {
    char *p, title[100], *pt;
    int cl;

    printf("Content-type: text/html\r\n", 10, 10);

    path=getenv("PATH_INFO");

    /* pick out the page title from the environment variable */
    pt=path;
    cl=strlen(path);
    pt=pt+cl-1;
    while (*pt != '/') pt--;
    pt++;
    strcpy(title, pt);
    plustospace(title);

    printf("<TITLE>Child page</TITLE>");
    printf("<H1>");
    printf("$s", title);
    printf("</H1>");
    header(path);
    ftw (path, profile, 30);
    tailer(path);
}
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>
#include <string.h>

int dirfile (char *fname, struct stat *statptr, int flag);
int prfile (char *fname, struct stat *statptr, int flag);
void header(char *path_info);
void tailer(char *path_info);
void plustospace(char *str);

char *path;
int file_number;

main(int argc, char *argv[]){
char *page_name, buf[500], *p, tmp_page_title[500], page_title[500];
int cl;

    printf("Content-type: text/html; charset=", 10, 10);
    page_name=getenv("PATH_INFO");
    path=getenv("PATH_INFO");
    file_number=-1;
    ftw(page_name, dirfile, 30);
    if (file_number > 0) {
        printf("<H1> The page is not empty.<P>\n");
        printf("Please press the back button to go back.<H1><P>\n");
        if (file_number == 0) {
            sprintf(buf, "rmdir %s", page_name);
            system(buf);
            p=page_name;
            cl=strlen(page_name);
            p=p+cl-1;
            while(*p != '/') p--;
            *p=NULL;
            p--;
            while(*p != '/') p--;
            p++;
            strcpy(page_title, p);
            plustospace(page_title);
            printf("Child page</TITLE>\n");
            printf("<H1>\n");
            printf("%s", page_title);
            printf("<H1>\n");
            header(page_name);
            /*printf("page_name is %s\n", page_name);
            printf("path is %s\n", path);*/
            ftw (path, prfile, 30);
            tailer(path);
    }
int dirfile (char *fname, struct stat *statptr, int flag)
    file_number++;
    return(0);
}
Form for Creating New Hyper-Link

/* This handles creating a new hyper-link */

#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[]) {
    char *path;

    printf("Content-type: text/html%c",10,10);

    path=getenv("PATH_INFO");

    printf("<TITLE>Creating Hyper-link</TITLE>");
    printf("<H1> Hyper-Link creating page</H1>");
    printf("<HR>");
    printf("<FORM METHOD="POST" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/create_link%s">", path);
    printf("Put the anchor into the following area: <P>");
    printf("<INPUT TYPE="text" name="anchor"<P>");
    printf("Put the URL into the following area:<P>");
    printf("<TEXTAREA ROWS=5 COLS=50 NAME="text"</TEXTAREA>");
    printf("<P>");
    printf("Put your name into the following area:<P>");
    printf("<INPUT TYPE="text" size="20" name="wname"">");
    printf("<P>");

    printf("<INPUT type="submit" value="create link">");
    printf("<INPUT TYPE="reset" value="Clear form">");
    printf("</FORM>");
}
**CGI Script for Creating New Hyper-Link**

/* This program is used to create hyper-link to a page */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>
#include <unistd.h>

#define MAX_ENTRIES 100

typedef struct {
    char *name;
    char *val;
} entry;

char *makeword(char *line, char *stop);
char *fmakeword(FILE *f, char *stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plastospace(char *str);
int profile(char *fname, struct stat *statptr, int flag);
void header(char *path_info);
void tailer(char *path_info);

char *path;
char buf[2000];

int main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x, m = 0;
    int cl, random;
    char ch, buf1[500], buf2[500], tmp_page_title[500], *page_title, *s;
    int fd;
    time_t now;
    srand((int) time(NULL));

    printf("Content-type: text/html%c\r\n", 10, 10);

    cl = atoi(getenv("CONTENT_LENGTH"));
    path=getenv("PATH_INFO");

    for(x=0; cl && (!feof(stdin)); x++) {
        m=x;
        entries[x].val = fmakeword(stdin, '"', &cl);
        plastospace(entries[x].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val, '"');
    }

    /* create a random name for the new file */
    chdir (path);
    random=rand();
    sprintf(buf1, "%d", random);
    fd=open(buf1, O_CREAT|O_WRONLY);
/* put contents to the file */
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="<A HREF="";
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="entries[1].val;
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="";
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s=entries[0].val;
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="</A>";
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));

/* attach date & author's name to the end of the file */
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s=entries[2].val;
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="/ "
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
time(&now);
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="</I><P">
strncpy(buf, s);
encry(0);
write(fd, buf, strlen(s));

/* change file mode */
sprintf(buf2, "chmod a+rwx %s", buf1);
system(buf2);

/* get the title for that page */
cl=strlen(path);
strncpy(tmp_page_title, path);
page_title=tmp_page_title;
page_title=page_title+cl-1;
while ( (*page_title="/" ) page_title--;
        page_title++;
        plustospace(page_title);

/* open each file on the page */
printf("<TITLE>Text page</TITLE>");
printf("<H1>%s</H1>", page_title);
header(path);
ftw(path, prfile, 30);
tailer(path);
}

cry(int index){
    index=0;
    while (buf[index] != '\0') {
        buf[index]=~buf[index];
        index++;
    }
    buf[index]= '\0';
}
Form for Adding New Text

/* This handles generating a form for putting new text to the page */
/* If user chooses using HTML mode, they can also create hyperlinks. */

#include <stdio.h>
#include <stdlib.h>

#define MAX_ENTRIES 1000

main(int argc, char *argv[]) {
    char *path;

    printf("Content-type: text/html\r\n",10,10);

    path=getenv("PATH_INFO");

    printf("<TITLE>Adding Text</TITLE>");
    printf("<H1> Text adding page</H1>");
    printf("<HR>");
    printf("<FORM METHOD="POST" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/add_text\s">", path);
    printf("<DL>");
    printf("<DT><B>Text style: </B></DT>");
    printf("<input type="radio" name="textstyle" value="normal" checked>Normal">
    printf("<input type="radio" name="textstyle" value="italic">
    &lt;i&gt;Italic&lt;/i&gt;");
    printf("<input type="radio" name="textstyle" value="bold">
    &lt;b&gt;Bold&lt;/b&gt;");
    printf("<input type="radio" name="textstyle" value="html">
    HTML code
    printf("</DL>");
    printf("Put the title into the following area: &lt;p&gt;");
    printf("<INPUT TYPE="text"
    name="title"><p>");

    printf("Put your name into the following area:&lt;p&gt;");
    printf("<INPUT TYPE="text" size="20"
    name="wname">";
    printf("&lt;p&gt;");

    printf("Put the content into the following form:&lt;p&gt;");
    printf("TEXTAREA ROWS=10 COLS=55 NAME="text"
    &lt;/TEXTAREA>";
    printf("&lt;p&gt;");

    printf("<INPUT type="submit" value="Add text">";
    printf("<INPUT TYPE="reset" value="Clear form">";
    printf("</FORM>");
}
CGI Script for Adding New Text

/* This handles adding new text to the page */
/* If user chooses using HTML mode, they can also create hyperlinks. */

#include <stdio.h>
#include <stdlib.h>

#define MAX_ENTRIES 1000

main(int argc, char *argv[]) {
    char *path;

    printf("Content-type: text/html\r\n\r\n", 10, 10);

    path=getenv("PATH_INFO");

    printf("<TITLE>Adding Text</TITLE>");
    printf("<H1> Text adding page</H1>");
    printf("<HR>");
    printf("<FORM METHOD="POST" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/add_text.pl">", path);
    printf("<DL>");
    printf("<DT><B>Select one text style: </B>");
    printf("<INPUT type="radio" name="textstyle" value="normal" CHECKED>Normal"");
    printf("<INPUT type="radio" name="textstyle" value="italic" >Italic</I>");
    printf("<INPUT type="radio" name="textstyle" value="bold" >Bold</B>");
    printf("<INPUT type="radio" name="textstyle" value="html" >HTML code";
    printf("</DL>");
    printf("Put the title into the following area: <P>");
    printf("<INPUT TYPE="text" name="title" >";
    printf("<P>");

    printf("Put your name into the following area:<P>");
    printf("<INPUT TYPE="text" size="20" name="wname" >";
    printf("<P>");

    printf("Put the content into the following form:<P>");
    printf("<TEXTAREA ROWS=10 COLS=55 NAME="text" ></TEXTAREA>");
    printf("<P>");

    printf("<INPUT type="submit" value="Add text" >";
    printf("<INPUT TYPE="reset" value="Clear form" >";
    printf("</FORM>");
}
Form for Appending Messages

/* This program is used to generate a form for appending message to the original file. */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>

int prfile (char *fname, struct stat *statptr, int flag);

main(int argc, char *argv[]){
    char *fname, buf[100], *p, ch;
    FILE *fd;

    fname=getenv("PATH_INFO");
    fd=fopen(fname, "r");
    printf("<H1>");
    printf("Appending text page");
    printf("</H1>");
    ch=getc(fd);
    while (!feof(fd)){
        putchar(ch);
        ch=getc(fd);
    }
    fclose(fd);
    printf("</HR>");
    printf("Use the form below to add new text.");
    printf("</HR>");
    printf("<FORM METHOD="post" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/info/append_text">", fname);

    printf("<DL>");
    printf("<DT><B>Text style:</B>");
    printf("<input type="radio" name="textstyle" value="normal"
    CHECKED>Normal"");
    printf("</input type="radio" name="textstyle" value="italic"
    ><I>Italic</I>");
    printf("<input type="radio" name="textstyle" value="bold"
    ><B>Bold</B>");
    printf("</DL>");

    printf("<TEXTAREA ROWS=10 COLS=55 NAME="Text">");
    printf("</TEXTAREA>");

    printf("<P>");

    printf("Put your name into the following area:<P>");
    printf("<INPUT TYPE="text" size="20" name="wname">");
    printf("</P>");

    printf("<INPUT TYPE="submit" VALUE="Append text">");
    printf("<INPUT TYPE="reset" VALUE="Clear form">");
    printf("</FORM>");
}
CGI Script for Appending Messages

/* This program appends text to the original file */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>
#include <unistd.h>

#define MAX_ENTRIES 100

typedef struct {
    char *name;
    char *val;
} entry;

char *makeword(char *line, char stop);
char *fmakeword(FILE *f, char stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plusisspace(char *str);
int printf (char *fname, struct stat *statptr, int flag);
void header(char *path_info);
void tailer(char *path_info);

char *path;
char buf[2000];

main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int i, m=0;
    int cl;
    int fd;
    time_t now;
    srand((int) time(NULL));

    printf("Content-type: text/html;charset","10,100);

    cl = atoi(getenv("CONTENT_LENGTH"));

    for(x=0; cl && (!feof(stdin)); x++) {
        m=x;
        entries[x].val = fmakeword(stdin, '\', &cl);
        plusisspace(entries[x].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val, '\n');
    }

    /* open the original file in the appending mode */
    fname=getenv("PATH_INFO");
    fd=open(fname, O_RDONLY | O_APPEND);

    /* lock the file until this action is complete */
    lockf(fd, F_LOCK, 0);
/* put contents into the file */
if (!strcmp(entries[0].val, "italicc")){
    s="<i>";
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
    s=entries[1].val;
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
    s="</i>";
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
} else if (!strcmp(entries[0].val, "boldd")){
    s="<B>";
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
    s=entries[1].val;
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
    s="</B>";
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
} else {
    s=entries[1].val;
    strcpy(buf, s);
    encry(buf);
    write(fd, buf, strlen(s));
}
/* attach date & author's name to the end of the file */
strcpy(buf, s);
encry(buf);
write(fd, buf, strlen(s));
strcpy(buf, s);
encry(buf);
write(fd, buf, strlen(s));
s="/ ";
strcpy(buf, s);
encry(buf);
write(fd, buf, strlen(s));
time(&now);
s=ctime(&now);
strcpy(buf, s);
encry(buf);
write(fd, buf, strlen(s));
s="</I><P>");
strcpy(buf, s);
encry(buf);
write(fd, buf, strlen(s));
close(fd);

/* get the title for that page */
c1=strlen(fname);
p=fname;
p=p+c1-1;
while (*p!='/') p--;  
*p=NULL;  
path=fname;  
p--;  
while (*p!='/') p--;  
p++;  
tmp_page_title=p;  
strcpy(page_title, tmp_page_title);  
plustospace(page_title);  

/* open each file in page1 directory */  
printf("<TITLE>Text page</TITLE>");  
printf("<H1>\$s</H1>", page_title);  
header(path);  
ftw (path, prfile, 30);  
tailer(path);  
}

encry(int index){  
index=0;  
while (buf[index] !='\0') {  
    buf[index]=~buf[index];  
    index++;  
}  
buf[index]= '\0';  
}
Form for Editing Text

/* This program is used to create a form to edit the original text. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>
#define BUF_SIZE 10000
int pfifile (char *fname, struct stat *statptr, int flag);
char *lock_path;
int boolean;
main(int argc, char *argv[]){
char *fname, tmpfname[500], *lock_fname, *p, ch;
FILE *fd, *lock_file;
int fp, t;
int length;
char lock_index[500][500];
int pifile(char *file_name, struct stat *statptr, int flag);
printf("Content-type: text/html;charset", 10, 10);
fname=getenv("PATH_INFO");
/* checking the lock directory to see if the file is being
edited by another user */
lock_path="/opt/local/WWW/Servers/NCSA/cgi-bin/wangd/Info/lock";
boolean=0;
strcpy(tmpfname, fname);
lock_fname=tmpfname;
bp=lock_fname;
bp=bp+strlen(lock_fname)-1;
while (*bp != '/') bp--;
lock_fname=bp;
strcat(lock_path, lock_fname);
ftw(lock_path, pifile, 1);
if (boolean == 1) {
printf("<H1>The file is being edited by another user. <P>");
printf("You cannot edit it now. Try later. <P>");
printf("Please use the back function to go back to the
previous page. </H1>");
}
else{
fp=fopen(lock_path, O_WRONLY | O_CREAT, 0666);
close(fp);
printf("<IMG SRC="/home/cs/misc/www/grads/"wangd/icon/Warning_btn.gif"/>");
printf("<BLINK><H1> Once you enter this page, you have to press
the edit button. It's forbidden to use the back function to go back
to the previous page. </BLINK></H1></BLINK><P>");
printf("<H1>");
printf("Test editing page");
printf("<H1>");
fd=fopen(fname, "r");
ch=getc(fd);
while (!feof(fd)){
putchar("ch");
ch=getc(fd);
}
fclose(fd);
printf("<HR>");
printf("Use the form below to edit the original text</B>");
printf("<HR>");
printf("<form method="post" action="http://www.cs.uwindsor.ca/cgi-bin/wang/Info/edit_text\&s">", fname);
printf("<dl>");
printf("<dt><b>Text style:</b></dt>");
printf("<input type="radio" name="textstyle" value="normal" checked>Normal" >/n");
printf("<input type="radio" name="textstyle" value="italic"><i>Italic</i></a>");
printf("<input type="radio" name="textstyle" value="bold"><b>Bold</b></a>");
printf("</dl>");
/* Allocate and clear buffer for file input */
buf=malloc(BUF_SIZE);
if (!buf) {
    printf("memory request failed\n");
    exit(1);
}
for (t=0; t<BUF_SIZE; t++) *(buf+t)='\0';
/* open the original file in read mode */
fp=fopen(fname, O_RDONLY);
length=read(fp, buf, BUF_SIZE);
/* omit the first six characters "<H4>--" */
bp=buf;
while (*bp != NULL) {
    *bp='*'
    bp=buf+6;
    printf(buf, "%s", bp_h);
    /* pick out the title */
    bp=memchr(buf, '-', 999);
    while (*bp) {
        if (((*bp)=='-') &&
            (*bp+1)=='-') &&
            (*bp+2)=='<') &&
            (*bp+3)=='/') &&
            (*bp+4)=='H') &&
            (*bp+5)=='4') &&
            (*bp+6)=='>' &&
            (*bp+7)=='<' &&
            (*bp+8)=='P') &&
            (*bp+9)=='>'
        } {
            bp_t=bp+10;
            *bp=NULL;
            title=buf;
        }
        else {
            do {
                bp++;
            } while (*bp && (*bp) != '-');
        }
    }
    printf("title is &lt;p"," title);
    /* put the title into the title area */
    printf("<title>");
    printf("<input type="text" name="title" value="", title>");
    /* pick out the original file contents without the appending message*/
    sprintf(buf, "%s", bp_t);
    bp=memchr(buf, '\', BUF_SIZE);
    while (*bp) {
        if (((*bp)=='/') &&
            (*bp+1)=='<') &&
            (*bp+2)=='I') &&
            (*bp+3)=='>'
        } {
            *bp=NULL;
        }
    }
    printf("</title>\n");
}
contents=buf;
bp_t=bp+4;
}
else {
do {
bp++;
} while (*bp & (*bp) != '\n');
}

/* put contents into the textarea */
printf("<P>Contents:<P>");
printf("<TEXTAREA ROWS=10 COLS=55 NAME="Text" >");
printf("%s", contents);
printf("</TEXTAREA>");
printf("<P>");
printf("Put your name into the following area:<P>");
printf("<INPUT TYPE="text" size="20" name="wname" >");
printf("<P>");
printf("<INPUT TYPE="submit" VALUE="Perform Edit" >");
printf("<INPUT TYPE="reset" VALUE="Reset form" >");
/* put the rest of the file into a hidden input which means can't be re-edited */
while (*bp_t != '\n') bp_t++;
bp=bp_t+4;
if (*bp != '\n') {
   appends=bp;
   printf("<P>");
   printf("The following text is the appending message to the original file:<P>");
   printf("<TEXTAREA ROWS=3 COLS=55 NAME="append" >");
   printf("%s", appends);
}
printf("</TEXTAREA>");
close (fp);
free(buf);
printf("</FORM>");
}

int plfile (char *file_name, struct stat *statptr, int flag)
{
   if (strcmp(file_name, lock_path) == 0) {
      boolen=1;
      return(1);
   }
   else return (0);
}
/* This program re-edits the original file */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <fcntl.h>
#include <unistd.h>

#define MAX_ENTRIES 100

typedef struct {
    char *name;
    char *val;
} entry;

char *makeword(char *line, char *stop);
char *fmakeword(FILE *f, char *stop, int *len);
char x2c(char *what);
void unescape_url(char *url);
void plusostospace(char *str);
int prfile (char *fname, struct stat *statptr, int flag);
void header(char *path_info);
void tailer(char *path_info);

char *path;
char buf[1000];

main(int argc, char *argv[]) {
    entry entries[MAX_ENTRIES];
    register int x,m=0;
    int cl;
    char *p, *fname, *tmp_page_title, page_title[500], tmp_fname[500];
    int fd;
    char *s, *lock_path, *lock_fname, rm_buf[500];
    time_t now;
    srand((int) time(NULL));

    printf("Content-type: text/html\c\c",10,10);

    cl = atoi(getenv("CONTENT_LENGTH");

    for(x=0; cl && (!feof(stdin)); x++) {
        m=x;
        entries[x].val = fmakeword(stdin, ',', &cl);
        plusostospace(entries[x].val);
        unescape_url(entries[x].val);
        entries[x].name = makeword(entries[x].val, '');
    }

    /* open the original file in read and write mode */
    fname=getenv("PATH_INFO");
    fd=open(fname, O_WRONLY);
/* put title to the file */
s=""H4""="";
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s=entries[1].val;
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s=""--""<H4><P>"";
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));

/* put contents to the file */
if (!strcmp(entries[0].val, "italico")) {
    s=""<I>"";
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
    s=entries[2].val;
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
    s=""</I>"";
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
} else if (!strcmp(entries[0].val, "bolded")) {
    s=""<B>"";
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
    s=entries[2].val;
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
    s=""</B>"";
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
} else {
    s=entries[2].val;
    strcpy(buf, s);
    encry(0);
    write(fd, buf, strlen(s));
}

/* attach new date & author's name to the end of the file */
s="" (""Changed by "";
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s=entries[3].val;
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s="" / ""
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
time(&now);..."
s = ctime(&now);
strcpy(buf, s);
encry(0);
write(fd, buf, strlen(s));
s = "</I><P>";
strcpy(buf, s);
encty(0);
write(fd, buf, strlen(s));
/* put the appending message to the file */
if (m == 4) {
    s = entries[i].val;
    strcpy(buf, s);
encty(0);
    write(fd, buf, strlen(s));
}
close(fd);
/* remove the file name from the lock directory, so the other
users can edit it. */
strcpy(tmp_fname, fname);
lock_fname = tmp_fname;
p = lock_fnume;
p = p + strlen(lock_fname) - 1;
while (*p != '/') p--;
lock_fname = p;
lock_path = "opt/local/www/Servers/NCSA/cgi-bin/wangd/Info/lock";
strcat(lock_path, lock_fname);
printf(rm_buf, "rm <%s", lock_path);
system(rm_buf);
/* get the title for that page */
c1 = strlen(fname);
p = fname;
p = p + c1 - 1;
while (*p == '/') p--;
*p = NULL;
path = fname;
p--;
while (*p == '/') p--;
p++;
tmp_page_title = p;
strcpy(page_title, tmp_page_title);
plustospace(page_title);
/* open each file in page1 directory */
printf("<TITLE>Test page</TITLE>");
printf("<H1>%s</H1>\n", page_title);
header(path);
ftw(path, prile, 30);
tailer(path);
}
encty(int index){
    index = 0;
    while (buf[index] != '\0') {
        buf[index] = "buf[index];
        index++;
    }
    buf[index] = '\0';}
Form for Deleting Text

/* This program is used to create a deleting text form. */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>

char *path;
char *lock_path;
int boolean;

main(int argc, char *argv[]){
  char *fname, buf[100], *p, ch;
  FILE *fd;
  char tmp_fname[500], *lock_fname;
  int pfile(char *file_name, struct stat *statptr, int flag);

  printf("Content-type: text/html;charset=", 10, 10);
  
  fname=getenv("PATH_INFO");

  /* check if the file is being edited by another user */
  lock_path="/opt/local/WWW/Servers/NCSA/cgi-bin/wangd/Info/lock";
  boolean=0;
  strcpy(tmp_fname, fname);
  lock_fname=tmp_fname;
  p=lock_fname;
  while (*p != '/') p--;
  lock_fname=p;
  strcat(lock_path, lock_fname);
  ftw(lock_path, pfile, 1);
  if (boolean == 1) {
    printf("<H1> The file is being edited by another user. <P>");
    printf("<H1> You can not delete it. </H1><P>");
    printf("Please use the back function to go back to the previous page."<H1>" );
  } else {
    fd=fopen(fname, "r");
    printf("<H1>");
    printf("Text deleting page");
    printf("<H1>");
    ch=getc(fd);
    while (!feof(fd)){
      putchar(ch);
      ch=getc(fd);
      fclose(fd);
      printf("<HR>");
    
    printf("<BODY TEXT="#FF00FF">");
    printf("<FORM METHOD="POST" ACTION="http://www.cs.uwindsor.ca/cgi-bin/wangd/Info/delete_text%&gt;", fname);
    printf("<B>Operation: </B><P>");
    printf("<INPUT TYPE="submit" VALUE="Delete this item">")
    printf("<P>);
    printf("Or Use the <B>back function</B> to go to the previous page");
printf("</FORM>");
printf("</BODY>");
}
}

int plfile (char *file_name, struct stat *statptr, int flag)
{
    if (strcmp(file_name, lock_path) == 0) {
        boolean=1;
        return(1);
    }
    else return (0);
}
CGI Script for Deleting Text

/* This program is used to delete the text */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>

int prfile (char *fname, struct stat *statptr, int flag);
void header(char *path_info);
void tailer(char *path_info);

cchar *path;
char *lock_path;
int boolean;

main(int argc, char *argv[]){
char *fname, buf[500], *p, *tmp_page_title, page_title[500],
tmp_fname[500], *lock_fname;
int cl;
FILE *fd;

printf("Content-type: text/html\c\c", 10, 10);

fname = getenv("PATH_INFO");
sprintf(buf, "rm %s", fname);
system(buf);

/* get the title for that page */
c1 = strlen(fname);
p = fname;
p = p + c1 - 1;
while (*p != '/') p--;
*p = NULL;
path fname;
p --;
while (*p != '/') p--;
p ++;
tmp_page_title = p;
strcpy(page_title, tmp_page_title);
plustospace(page_title);

/* open each file in page1 directory */
printf("<TITLE>Text page</TITLE>");
printf("<H1>%s</H1", page_title);
header(path);
ftw(path, prfile, 30);
tailer(path);
}
CGI Script for Creating Index Tree

/* This program presents the index directory tree of the IS */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <ftw.h>
#include <string.h>

#define SPACE 3

void plutospace(char *str);

typedef struct {
    char node[800];
    int length;
    int child;
} directory;

int dfree (char *fname, struct stat *statptr, int flag);
directory tree[800];
char *path;/* tree[500][50000]; */
int i, max;
int y;

main(int argc, char *argv[]) {
    printf("Content-type: text/html;charset=ISO-8859-1\n\n");
    path="/opt/local/WWW/Servers/NCSA/cgi-bin/wangq/Info/Top+Page";
    printf("<TITLE>Index Page</TITLE>\n\n");
    printf("<H1>Directory Index</H1>\n\n");
    i=0;
    max=0;
    ftw (path, dfree, 30);
    printf("<BR>\n\n");
    printf("<PRE>\n\n");
    for (y=0; y<i; y++) plutospace(tree[y].node);
    printf("%s<BR>,tree[y].node);\n\n");
    printf("</PRE>\n\n");

    int dfree (char *fname, struct stat *statptr, int flag){
    char buf1[800], buf2[800], *c;
    int cl, x, n, index;
    int count;

    if (flag==FTW_D) {
        sprintf(buf1, fname);
        info_path=buf1;
        p=info_path;
        cl=strlen(info_path);
        p+=cl-1;
        while (*p != '/') {
            name=buf2;
            if (p=\n\n\n2
p--; 
while (*p != '/') p--; 
parent = p;

p = fname; 
index = 0; 
for (count = 0; count < cl; count++) {
    if (*p == '/') index++; 
p++; }
index = index - 9;

if ((strcmp(name, "/Top+Page") == 0) && index <= 0) {
    max = strlen(name); 
sprintf(tree[i].node, name); 
tree[i].length = strlen(name); 
tree[i].child = 0;
} 
else if ((strcmp(parent, tree[i].node) == 0)) {
    tree[i].child ++;
    sprintf(tree[i+1].node, "."); 
    sprintf(tree[i+2].node, "|"); 
    sprintf(tree[i+3].node, "|"); 
    i = i + 4;
    sprintf(tree[i].node, name); 
tree[i].length = strlen(name); 
tree[i].child = 0;
    if (max < strlen(name)) max = strlen(name);
} 
else {
    x = index - 1; 
x = x * 4;

    if ((x < i) && tree[i].child == 0) {
        tree[i].child ++;
        sprintf(tree[i+1].node, " "); 
        sprintf(tree[i+2].node, " "); 
        sprintf(tree[i+3].node, " "); 
        sprintf(tree[i+4].node, " "); 
        count = max - tree[i].length - 2;
        for (n = 0; n <= count; n++) { 
            strcat(tree[i+1].node, " "); 
            strcat(tree[i+2].node, " "); 
            strcat(tree[i+3].node, " "); 
            strcat(tree[i+4].node, " ");
        }
        strcat(tree[i+1].node, " "); 
        strcat(tree[i+2].node, " |"); 
        strcat(tree[i+3].node, " |"); 
        i = i + 4;
        strcat(tree[i].node, name); 
tree[i].length = strlen(name); 
tree[i].child = 0;
        if (max < strlen(tree[i].node)) max = strlen(tree[i].node);
    } 
    else if (tree[x].child == 0) {
        tree[x].child ++;
        count = max - (tree[x].length - strlen(tree[x+1].node)) - 1;
        for (n = 0; n <= count; n++) { 
            strcat(tree[x+1].node, " "); 
            strcat(tree[x+2].node, " "); 
            strcat(tree[x+3].node, " ");
        }
        strcat(tree[x+1].node, " "); 
        strcat(tree[x+2].node, " |"); 
        strcat(tree[x+3].node, " |"); 
        strcat(tree[x+4].node, " ");
        count = count - tree[x+4].length + 1;
        for (n = 0; n <= count; n++) { 
            strcat(tree[x+4].node, " "); 
            strcat(tree[x+4].node, name);
}
tree[x+4].child=0;
tree[x+4].length=strlen(name);
if (max < strlen(tree[x+4].node)) max=strlen(tree[x+4].node);
}

else {
    tree[x].child++;
    count=max+strlen(tree[x+1].node)+SPACE;
    for (n=0; n<count; n++) {strcat(tree[x+1].node, "_");
        strcat(tree[x+2].node, " ");
        strcat(tree[x+3].node, " ");
    }
    strcat(tree[x+1].node, " ");
    strcat(tree[x+2].node, "|" );
    strcat(tree[x+3].node, "|" );
    count=max-strlen(tree[x+4].node)+SPACE;
    for (n=0; n<count; n++) strcat(tree[x+4].node, " ");
    strcat(tree[x+4].node, name);
    if (max < strlen(tree[x+4].node)) max=strlen(tree[x+4].node);
    tree[x+4].child=0;
    tree[x+4].length=strlen(name);
}

}

return(0);
}
BIBLIOGRAPHY


VITA AUCTORIS

May Wang was born in 1967 in Dalian, P. R. China. She graduated from The 24th High School of Dalian in 1985. From there she went on to the Changsha Institute of Technology where she obtained a B. Sc. in Computer Science in 1989. She is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in the Spring of 1996.