CSC400W: Introduction to Digital Libraries

Second Semester 2003

Course Description

Digital libraries are a specific type of information system that attempts to organise data and provide relevant services (such as search engines and directories) to users seeking information. This course will look at the various policy and technical issues involved in building digital libraries, as well as relevant standards and practices that are broadly applicable to networked system design in general and specifically Web application design.

Instruction

Instructor: Hussein Suleman (*hussein@cs.uct.ac.za*)
Office hours: MWF 2-3pm
Office: 310 Computer Science Building

Teaching Assistant: Simon Perkins (*sperkins@cs.uct.ac.za*)

Lectures: 24, as per calendar (approximately 4 will be class discussions and 1 will be a field trip)

Credits: 3

Textbook/Notes

There is no prescribed textbook, but the following is recommended:

> Digital Libraries
> *by William Y. Arms*
> Available online at http://www.cs.cornell.edu/wya/DigLib/

Class notes (copies of slides) will be made available at the Computer Science reception.

Course Outline

- Definitions and examples of DLs
- DL models, data and services
- Markup languages, hypertext, XML, SAX, DOM
- Data definition, schema, metadata, DC
- Data transformation, XSLT

- Repositories and archives, identifiers, RAP, SODA, FEDORA
- Searching
- DL as Web application, Greenstone, EPrints
- DL as Web service, search protocols, SOAP
- Data transfer and repository access protocols, RAP
- Open Archives Initiative and OAI-PMH
- ODL, OpenDLib, and other component models
- Orchestration and composition, WS-Coordination, 5SL
- Distributed and hierarchical systems
- The library perspective, MARC, Z39.50
- Field trip: library
- IP Rights, DRM
- Interfaces and usability, portals, mobile devices, community support
- Preservation, LOCKSS
- Open access, BOAI, OAIS, economics and business models

Grade Weighting

4 Programming Assignments, each 2 weeks in length (65%)

Class participation (10%)

Final Exam (25%), take home

DP Requirement: 50% on the average of programming assignments

---

Collaboration

All assignments, tutorials and exams must be done strictly on an individual basis. Design and coding of programming assignments must be done strictly on an individual basis. It is acceptable to discuss with classmates the question but not the solution (neither algorithmically nor programmatically). In no way should the individual statements of a program or the steps leading to the solution of the problem be discussed with or shown to anyone except the teaching assistants, the instructor, and the tutors. Any discussion of your program source code must be limited to these people.

Always give credit for work that is not entirely your own (e.g., parts of programs or algorithms derived from a book).

Assignments and Grading Policies

General

All graded work (whether in paper or email format) must be kept until the end of the semester.

In general, queries about grades MUST be made within a week of graded work being returned. No

queries about any grades will be entertained after the final examination.

ALL students will be expected to complete ALL assigned work (assignments and examinations). If you miss ANY assigned work with a legitimate reason, speak to the instructor within a week or as soon as possible thereafter. Note that there are few legitimate reasons that will be accepted - these include condoned absences for recognised religious observances and hospitalization or serious illness (with certificate).

Examinations

The final examination will be take-home (and also open-book and open-notes). It will be cumulative (i.e. covering all material of the course).

Answers must be typeset using a word processor and printed out on paper. No handwritten work will be accepted.

A submission date will be set when the exam is provided. No late submissions will be accepted.

Programming Assignment

All questions for assignments, along with all related requisites, will be available online on the course website.

Grading of programming assignments will be based on the following criteria (an approximate marking scheme is indicated in brackets) :

1. Correctness of program [50%].
2. Output from program that adequately demonstrates correctness [10%].
3. Documentation, internal and external, included as appropriate [15%].
4. Efficient use of algorithms and appropriate data structures [10%].
5. Stress - program must function correctly under all and/or extreme and unusual combinations of input [10%].
6. Creativity - credit for innovation in interface, implementation, style, etc [5%].

All programming assignments are due at the date and time specified. No late submissions will be accepted unless a general extension has been granted. In very rare instances, exceptions may be made at the discretion of the instructor on an individual basis. Requests for extensions must be made at least 24 hours in advance of the due date.

All programming projects will be submitted electronically. The online submission system used to receive your assignments will provide the official timestamp used to determine whether a program is on time.

NOTE: You are not allowed to use the **goto** statement for any programming done in this course !

What constitutes Output ?

Output refers to screen dumps of the output generated by your assignment, also indicating input where relevant. These are typically cut-and-pasted into word-processor documents. The basic reason for them is so that the person grading your assignment need not have to run the program to see exactly what the output looks like.

Screen dumps are usually acquired on Windows systems by pressing the PrtScrn button on the keyboard and then "pasting" the clipboard's contents into a program such as "Microsoft Word". A similar cut-and-paste operation can be used on *nix systems. The file containing your output must be included with your submission.

A cleaner alternative is to structure programs such that input is obtained from one file and output is sent to another - both files should then be included with your submission. Where this is not possible the above solution should be adopted.

In all cases, all files in your submission must be appropriately named, with a README file provided if necessary, so the purpose of each file is obvious to markers.

What constitutes Creativity ?

Creativity is any substantial improvement beyond the basic solution - it can be applied to any part of the project. For example, the following are relevant to data structures assignments:

1. User Interaction
    - It may be useful to present the user with options to either test the program using internal tests or an interactive interface.
    - Work around limitations in the program. For example, if the program asks for lines of input and quits when it sees "X", devise a special syntax (called an escape sequence) to allow the user to type in "X" without the program exitting. Hypothetically, if the user enters $X the program interprets it as X, if the user enters $$ the program interprets it as $ and if the user enters X the program exits.
2. Visualization
    - A representation of how data is actually being stored in a data structure, by specific position and value (this would even help greatly with debugging) - this could be accomplished with specialized data access routines for output formatting.
3. Testing
    - Simulating real-world conditions for input by making some assumptions about the distribution and rate of transactions, then simulating using random number generators.
    - Exhaustive automatic testing - go through many (or every) possible scenarios (up to some time limit). For example, assign "add" and "remove" to a binary variable - then generate all possible strings of operations - test the program for each case, and test the results automatically in your main program to make sure they are what was expected.
    - Develop a syntax for file-based testing and use this as an option - eg. "enter X" and "retrieve".
4. Efficiency
    - Minimize the number of allocations of memory blocks by reusing deleted blocks.
    - Use algorithms with better time or space efficiency than the standard ones.

Ask yourself these questions ...

- How can I make the interface more natural ?
- How can I make the program run faster ?
- How can I use less memory/disk space ?
- Have I thoroughly tested my program ? Will my program survive real-world tests ? Will my program survive worst-case scenario tests ?

Equipment and Programming Language

It is the responsibility of the student to submit programs that will successfully compile and execute on the standard departmental laboratory computers.

Information Dissemination and Communication

This is a lecture course. While attendance at lectures is not mandatory after the first day, all graded work (exams, assignments) will be based on material covered in the lectures.

Information will be added on a regular basis to the class website. All students will be expected to consult the website on a daily (Monday-Friday) basis for updates on assignments, tutorials, grades, hints, deadlines, etc.

All discussion of a general nature will take place on the class newsgroup. At the discretion of the instructor or TA, responses to individual email messages may be posted in the newsgroup if the responses are relevant to all students.

Please refrain from posting anything of the following nature on the newsgroup, or in other fora set up for the class, as it may form the basis for a violation of university policy and/or a legal transgression:

- sexist, racist or otherwise discriminatory comments
- segments of program code (other than something provided by the instructor)
- solutions to graded work (or part thereof) before the deadline for submission
- flame wars
- illegal material

Last updated : 25 July 2003 12.23pm