Field-of-Study Course Proposal

Part A.

Proposed field of study: Symbolic Reasoning
Course number: CMSC 105
Full course title: Elementary Programming

Offerings of the course will have a topical focus that may change from offering to offering, and thus we anticipate qualifying the title to identify the focus. So, for example, the course might appear in the course listings for a semester as “Elementary Programming: Robotics applications.”

Catalog description:
Solving problems by writing computer programs. Introduction to computer architecture. Emphasis on symbolic reasoning using examples from a particular computing context. For non-majors. Not open to students who have completed any Computer Science course that fulfills major requirements. 3 sem. hrs.

Prerequisites: none
Hours of credit: 3 semester hours
Estimate of student enrollment: 20 to 40 students per semester.
By whom and when the course will be offered:
The course will be offered initially by Lewis Barnett, though it could be taught by any computer science faculty member. We anticipate offering the course every semester contingent upon demand and staff availability.

Staffing implications
Demand for CMSC 150, which peaked at four sections per semester in the early part of this decade when interest in computer science as a major was very strong, have declined substantially. This course is intended to replace some of those sections, and thus should be staffing neutral.

Adequacy of library, technology, and other resources
The resources required for this course are similar to the needs for CMSC 150, and since we have handled much higher enrollments in that course with current facilities, there should be no difficulties supporting the new course.

Relation to existing courses and curricula: none

Departmental approval:
The course was approved by the computer science faculty on 4/8/2005.

Purpose and rationale:
Our department offers several FSSR courses designed for students who do not intend to continue study in mathematics or computer science (e.g. Math 102 – Problem Solving using Finite Mathematics; Math 103 – Introduction to Simulation; CMSC 101 – Minds and Machines; etc.). CMSC 105 is another course of this nature. The course will provide students with an introduction to solving problems using computer programs as a tool. We anticipate that the programming examples and assignments in each offering of the course will follow a theme determined by the instructor. Two possible examples of themes under discussion for this course are Web Scripting (programs that are part of an interactive web site) and Robotics.

Contact Person: Lewis Barnett, x8091, lbarnett@richmond.edu
Part B:

Brief outline:

Students in the course will solve problems using from one to three programming systems. The problem solving will include “translating problems”\(^1\) into computer programs from problem statements that do not make clear the particular solution method to be used. In addition, the course will cover relevant background information, such as the rudiments of computer architecture and other aspects of the nature of computing systems and of computer science. This background information will also include programming language syntax and semantics, so students can recognize and correct syntax errors and can precisely predict the output of the kind of programs they write, given specified inputs. However, the central theme of the course will be the kind of problem solving by students that expresses a solution in the form of a computer program.

For further details, see the attached statement *Information to the teacher* and sample syllabus from a prior offering of the course.

---

\(^1\) All quotes are from the symbolic reasoning field of study description.
**Information to the teacher**

The programming system must be nontrivial. For example, it cannot be a word processing system, spreadsheet system, or web page design in HTML. The scope of the programming by students must include emulating, at least at a simple level, key elements of problem solving actually carried out by computer software professionals. Examples of such problem solving include (but are not restricted to):

- developing programs that use looping to process input data to yield summary statistics,
- developing programs that enable a simulated robot to navigate within an environment,
- developing graphical user interfaces that actively respond to user actions by executing appropriate subprograms, and
- developing expert systems using symbolic facts and symbolic transformation rules.

Examples of programming systems that may be covered in the course include (but are not restricted to):

- assembly language,
- Karel the Robot or JKarelRobot,
- NQC (Not Quite C – a simplified version of C that runs on Lego Mindstorms),
- Prolog,
- Python,
- PHP,
- Lisp,
- Scheme,
- Visual Basic,
- C++,
- C# and
- Java.

In view of the course objective, and since this course is not a prerequisite for another course, the choice of programming systems is up to the individual teacher, as long as the systems satisfy the requirements stated here. In choosing a programming system, teachers may attempt to optimize such goals as one or more of the following:

- student interest,
- long-term benefit to students,
- challenge to the students,
- real-world applicability, and
- student success in achieving problem solving.

The teacher's personal excitement about a programming system is also an important factor, since students are apt to find such excitement contagious. Teachers are encouraged to share their successes, failures, and evolving opinions with other teachers, so the course can be enriched through time by these collective experiences.
In satisfying the symbolic reasoning curriculum requirement, students taking the course should experience the advantages of abstraction, such as programming using subprograms and stepwise refinement. Thus at least one programming system at a higher level than assembly language must be used. Stepwise refinement is an example of one of the “underlying principles that govern the application” of the rules of the system. Another “underlying principle” is the fact that the boolean condition on a while loop is false upon termination of the loop.

Since the primary course emphasis is on problem solving and not on mastering the syntax of a particular programming language, student use of a tool such as a syntax-directed editor is not inappropriate, as long as the students nonetheless “understand consistent rules by which the information relevant to a problem may be processed in order to obtain a solution”.

The symbolic reasoning requirement also mentions “judging both the appropriateness of known solution methods to a particular problem and the quality and reasonableness of the solution obtained.” This part of the requirement has been interpreted to mean that limitations of the symbolic system must be considered. Limitations of computer programming that can be considered include the need for understanding the theory behind a problem well enough to formulate an algorithm, the need for discretizing data, the difficulty of ensuring that software is correct, and possibly the intractability of certain problems, both because of combinatorial explosion and because of provably inherent intractability, such as the unsolvability of the halting problem.

The field of study requirements are intended to ensure Arts and Sciences students actually experience a wide variety of fields of study. Thus, in order for such a course to be successful, it should not only be challenging but one that the vast majority of Arts and Sciences students, if they work diligently, can pass.
CS105 - Problem Solving via Computer Programming

Professor

Lewis Barnett      Office                                  Home
       Jepson Hall, 226                                    (804) 741-3116
       (804) 289-8091                                      (804) 289-8091
email: lbarnett@richmond.edu

Office hours: 2:30pm - 3:30pm MW, 9:00am - 10:00am TTH, or by appointment

Prerequisites

None.

Statement of Course Objective

(This is a detailed statement on how CMSC 105 satisfies the Fields of Study: Symbolic Reasoning requirement.) Students will solve problems using one or more programming systems. The problem solving will include translating problems into computer programs from problem statements that do not make clear the particular solution method to be used. In addition, the course will cover relevant background information, such as the rudiments of computer architecture and other aspects of the nature of computing systems and of computer science. This background will also include programming language syntax and semantics, so that students can recognize and correct syntax errors and can precisely predict the output of the kind of programs they write, given specified inputs. However, the central theme of the course will be the kind of problem solving by students that expresses a solution in the form of a computer program.

Comments

From start to finish, this course is about writing computer programs to solve problems. Even so, no programming background is assumed -- we will be starting from scratch in this course. You will find many similarities between the organization of this course and Math courses. You will be doing problem sets for homework, and the main components of tests will be problem solving exercises. We will start the semester by establishing the notation we will use (in this context, "notation" means the programming language in which we will write our programs), then begin using the notation to solve problems which will increase in complexity as the semester progresses. You will find that later material builds on earlier material, making it very important to keep up with what is going on in class.

The main computer equipment we will use this semester are the Macintoshes located in the Public Lab on the ground floor of Jepson Hall. The software we will be using is free, so if you have a computer of your own that you would like to use for the assignments, you will be able to do so. Similar software is also available for IBM PCs -- if you have a PC, and would like to use it for your homework assignments, see me.

As our "notation" this semester, we will be using the Scheme programming language. Scheme is a simplified version of the LISP programming language, which is widely used in Artificial Intelligence and for rapid prototyping of software systems. It is very different from languages
like BASIC or PASCAL both in terms of the syntax of the language and in terms of how you think about creating solutions to problems. As a result, you may in fact discover that having no previous experience in programming is an advantage rather than a disadvantage in this course.

If Scheme is so different from more "normal" programming languages, why did I choose it for this course? There are several reasons, some of which tie in directly with the objectives of this course:

1. Problem solving is the focus of the course, not learning syntax. Languages like PASCAL have a very extensive set of syntactic constructs. To fully learn just the syntax of the language might take up to 50% of class time. This makes sense if you are going to be using PASCAL or other PASCAL-like languages in other courses, but not in the context of a course like this one. Scheme has a very simple, regular syntactic structure, allowing us to very rapidly move from the "learning notation" stage of the course to the "solving problems" stage.

2. We want to look at the important ideas in problem solving using computers, such as modular program structure and recursive program design (these terms will be explained later, I promise!). These ideas are embodied directly in the Scheme language in a more obvious way than in the other languages mentioned.

3. We want as much "hands on" work with computers as possible. As mentioned earlier, you can get the software that we will be using this semester for your own computer if you have one, no matter what kind of computer you have. I hope this will allow you to try out the ideas that we discuss in class very easily on the computer while you work on your homework assignments.

Readings


CS105 Handout Packet

Attendance

Each unexcused absence will result in a 1 point deduction from your final average. Excuses need not be written. When possible, please inform me of absences beforehand.

Grading Policy

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>10%</td>
</tr>
<tr>
<td>Projects</td>
<td>20%</td>
</tr>
<tr>
<td>Quizzes (4)</td>
<td>40%</td>
</tr>
<tr>
<td>Final (cumulative)</td>
<td>30%</td>
</tr>
</tbody>
</table>

Grades will be assigned on a 10 point scale. Makeup quizzes will not be given; in the event that a quiz is missed, the final exam grade will be substituted for the missed quiz in calculating the final grade for the course. The lowest quiz grade will be dropped. Missed homework and project assignments will receive a grade of 0.
Assignments

There will be a number of homework problem sets (approximately one per week) which will be graded and returned. There will also be several larger programming projects. Late assignments will be grudgingly accepted with a letter grade per day penalty.

The Honor Code

All quizzes and programming projects will be pledged under the honor code, unless the assignment explicitly states otherwise. Homework assignments are not pledged. If you are so inclined, feel free to work on the assignments in small groups. However, when you write up the problem set, please use your own words. Two people making longhand copies of one person's work is not acceptable.

Course Outline

I plan to cover the following topics this semester. The ordering may be rearranged somewhat. Each topic, to the extent possible, will be illustrated by several example programs. Quizzes will be scheduled approximately every fifth class meeting.

- Introduction to the Macintosh
- The basic structure of computers
- Introduction to the MacGambit Scheme Interpreter
- Notation -- The Syntax of the Scheme Programming language
  - Atoms, symbols and lists
  - List Manipulation -- putting things together and taking them apart
  - Arithmetic -- Infix notation for arithmetic expressions
  - Making choices -- cond expressions
  - Creating functions
- Program structure
  - Using functions to construct modular programs
  - Recursion
  - Iteration
- Interactive Programs
- Vectors
- Using files
- Case Studies of Applications Developed with Scheme

CS105-1 (12:45) Final Exam       Thurs., Dec. 12, 2:00 -- 5:00pm
CS105-2 (2:15) Final Exam       Tues., Dec. 17, 9:00am -- 12:00 noon