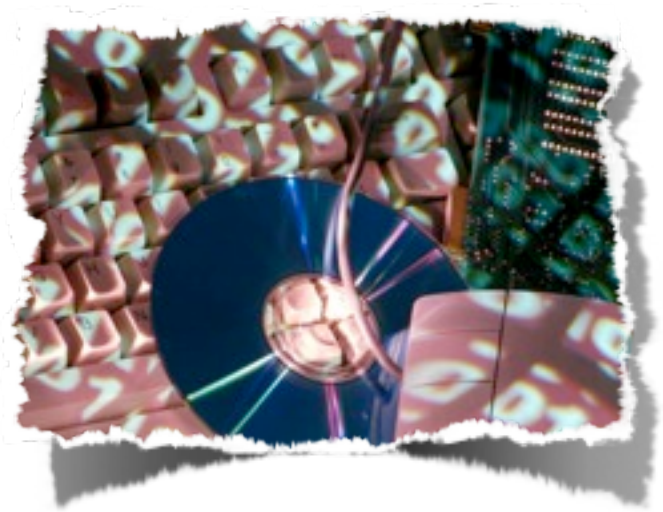
Design and Implementation of Programming Languages



CSC301 - Fall 2012

Online Syllabus

General Course Information

Time & Location: *Online!*

Instructor Information:

Instructor: Dr. Timothy Henry, PMP
Email: thenry@cs.uri.edu
Office: Room 257, Tyler Hall
Phone: 401-874-2701

Office Hrs: Tuesday, 10:00 AM- 11:00 AM and by appointment

Prerequisites: CSC212 - Data Structures and Abstractions

Textbook: *Concepts of Programming Languages, 9th Ed.;* Sebesta, Robert W.; Addison-Wesley & Associates; 2009; ISBN: 978-0-13-607347-5

Software: Racket (Scheme) and SWI-Prolog

Webpage: *URI Sakai Portal* - `CSC301-12F`

Course Overview

A programming language is the programmer's principle interface with the computer. More than just knowing how to program in a single language, programmers need to understand the different styles of programming promoted by different languages. In your professional life, you will be working with many different languages and styles at once. You will encounter many different languages over the course of your career. Understanding the variety of programming languages and the design tradeoffs between the different programming paradigms makes it much easier to master new languages quickly. Understanding the pragmatic aspects of programming languages also requires a basic knowledge of programming language translation and runtime features such as storage allocation.

After completing this course, you will have the tools and techniques to

- improve your ability to develop effective algorithms,
- improve the use of your existing programming language,
- increase your vocabulary of useful programming constructs,
- allow a better choice of programming language,
- make it easier to learn a new language, and
- make it easier to design a new language.

This course is based on Course CS240s from the 2008 update to the 2001 ACM/IEEE Computing Curricula. It introduces the theory and practice of programming language translation. Topics include lexical analysis, parsing, symbol tables, declaration and storage management, code generation, and optimization techniques.

Course Web Site

Because this is an ONLINE course; significant responsibility falls on you, the student, to keep up with the work and not fall behind!

As you can expect, we will use the course web site for all aspects of this course. Students are expected to check the web site regularly for:

- General Course Announcements & Assignment Updates
- Quizzes, Assignment Submission and Grades
- Discussion Boards
- Course Calendar
- Reference Materials

Any updates to the assignment syllabus will be posted on the course web site. Therefore, if there is a difference between the assignment syllabus and the course web site, information from the course web site should be used. To ensure you receive the maximum credit for your work, follow any templates or guidelines that are provided.

Getting Help in this Course

Questions concerning the content of the course or projects should, as a rule, be directed to the appropriate course discussion forum. This allows your question to be answered by whoever is monitoring the discussion forum, and the answer can benefit all board readers. Please do not send email directly to the TA (teaching assistant) or Dr. Henry with technical questions.

We (the TA and Dr. Henry) are happy to answer questions during office hours and on the discussion forum. However, office hours and email are not intended as a replacement for the discussion forums. As a result, we will only see people during office hours who regularly participate in class. Due to our own work schedules, we are not online 24x7 so we may not respond to electronic questions instantly. If you cannot make it to scheduled office hours, feel free to make an appointment by e-mail or after class.

Grading Information

Material in any course is not learned or mastered simply by reading the material or watching videos. The student needs to spend the time doing the readings, discussing the issues with fellow students, and doing activities based on the course concepts.

The final grade will be based on individual grades received on homework and programming assignments, quizzes, and exams, group projects and class attendance and participation. Point values for assignments, quiz questions and exam problems are based on the level of effort and knowledge required to complete each. The *approximate* weighting for each area is as follows:

- 20% -- **Class and Discussion Participation**
- 20% -- **Homework Assignments**
- 20% -- **Programming Assignments**
- 20% -- **Mid-Term Exam**
- 20% -- **Final Exam**

Course Assignments and Student Responsibilities

Weekly Assignments and Course Organization

Each assignment week begins on a Sunday and ends on the following Sunday. In general, all assignments and discussions posts are due on either a Wednesday or a Sunday (at midnight). Here is a brief summary of the assignments we will have each week:

- **Readings** - one chapter from *Concepts of Programming Languages, 10th Ed*, Sebesta.
- **Video Lectures** - approximately 60 minutes of video lectures based on the readings. (PDF's of the lecture slides are also provided).
- **Homework** - a homework assignment based on each chapter.
- **Programming Assignments** - a programming assignment in either Racket or Prolog.
- **Online Discussions** - make at least three posts in the discussion forum for each week.

Reading Assignments *(not graded)*

Reading assignments should be completed as early in the week as possible. Reading the assigned pages in the textbook gives you an idea of the concepts that are the focus for the week. Each of the other assignments during the week help to explain the concepts in greater depth and give you experience applying them to actual problems. Therefore, since the readings are fundamental to activities for the week, they should be completed first.

For this course to be successful, you must engage in the material by doing the readings ahead of time, and then by participating in online discussions. You will be expected to actively participate by asking questions, joining in our discussions, etc. Note that a significant portion of your grade is attributed to class participation.

In addition to the textbook readings, there will occasionally be notes or slides that outline or expand on key concepts for the week. Reading or reviewing these can help you to understand and organize the material better.

Video Lectures *(not graded)*

The video lectures for this course present the material in a form different from the textbook so that you can better understand the concepts in each section. The videos are provided by the instructor and are in QuickTime and Windows Media Format. These emphasize important concepts and give examples similar to what you would see in a classroom lecture.

Weekly Topic Online Discussions *(graded)*

Graded Forums

Because this is an online course, the weekly discussions will form an integral part of your grade. To receive full credit for your participation in our discussions, you need to make at least three posts that add substance to the two weekly conversations.

Your posts should demonstrate you understand the materials assigned. Your responses might integrate multiple views and/or motivate other students to respond. You should provide evidence that you are reading the assigned materials and other students' postings and bring out interesting interpretations. Don't just demonstrate that you know the facts, but show that you are able to analyze them and handle conceptual ideas. Posts that only state, "I agree" or "That is an interesting idea" are OK in the forum, but they will not receive credit. Feel free to contribute from your personal experiences or your own research or interests.

Each week you need to make your first post to the discussion forums by Wednesday, midnight. Then contribute two additional posts to the forum by Sunday, midnight. Each of your posts can earn you up to 5 points for a maximum of 15 points per discussion forum. The higher quality your post, the more points you will earn. There will be a 3 point penalty for each forum in which you do not make your first post by Wednesday. *Posts will not be accepted for grading after the week for the discussion has passed*.

You can post more than three times in each forum, especially if you feel that your posts may not be of high quality or you have additional comments to make on the topic.

Homework Assignments *(graded)*

The purpose of homework is to reinforce the conceptual material from the textbook and are usually taken from the questions at the end of each chapter. Many questions are of a variety that could also appear on exams. Homework assignments that are submitted:

- on time are eligible for full credit based on the rubric for the assignment
- within a week of the due date will be penalized 25%
- more than a week late receive a 50% penalty

Programming Projects *(graded)*

This course has multiple a programming projects to complete. Each programming project is to be completed and submitted as your own work. If you need assistance, make a post on the course help forum or contact the TA or professor and set up a time to come in form help.

Each student will submit a solution that accurately represents his or her effort on the project. *Every programming assignment requires documentation (comments)*. The assignment page for each assignment has additional information on what is required for each programming project.

Programming projects that are submitted:

- on time are eligible for full credit based on the rubric for the project
- within a week of the due date will be penalized 25%
- more than a week late receive a 50% penalty

Exams *(graded)*

There will be two examinations given in this course: a mid-term exam and a final exam. Students are to submit their own work on each of the exams. If you need help or clarification on a problem or exercise, you are to make a post on the discussion forum or contact the TA or instructor.

- You are not permitted to get help from other students when completing exams.
- Late exams will be penalized the same as late homework assignments.
- The material in this course is cumulative, and so the final exam will be cumulative.

Q & A Discussion Forums *(not graded)*

Non-Graded Help Forums

Help forums in this course can be a valuable resource. As a computer science professional or other scientist, there are many times that discussion forums can give you information or insight to quickly solve problems you have run into (and ways to avoid problems in the future). Each of the assignments has a discussion forum for problems, issues and advice.

When you are stuck on an assignment or consistently run into the same problem, first check that project's forum to see if someone else had the same problem. If not, post your comment or problem description there. Use descriptive titles for your posts so that other students know what the post is about. Titles such as "I have a problem" are not acceptable. If you have solved a problem that you see someone else struggling with, feel free to post advice on how to work through the problem, but do not post assignment solutions! I will be monitoring the discussions to ensure that this rule is followed. In general, you should not need to post more than three or four lines of code.

Preliminary Assignment Syllabus

In addition to the readings listed below, each week will also have the following graded assignments: a homework assignment and a programming project,.

The topics below are subject to change based on the actual pace of the course. Please check the course web site for the most up-to-date information.

Language Theory and Language Categories			
Sept 2	Week #1	Chapter 1	Introduction to Studying Programming Languages
Sept 9	Week #2	Sections 2.4, 2.13, 2.15 - 2.20 Sections 15.1 - 15.6, 15.10 - 15.11	Important Language Categories and History Functional Programming Languages and Scheme
Sept 16	Week #3	Chapter 11	Abstract Data Types and Encapsulation
Sept 23	Week #4	Chapter 12	Support for Object-Oriented Programming
Sept 30	Week #5	Sections 3.1 - 3.3	Describing Syntax
Oct 7	Week #6	Sections 3.4 - 3.5 Sections 4.1 - 4.2	Describing Semantics Lexical and Syntax Analysis
Mid-Term Exam Due - Wednesday, Oct 17, 2012 (Date Subject to Change)			
Programming Language Concepts and Implementation			
Oct 14	Week #7	Chapter 5	Names, Binding and Scope
Oct 21	Week #8	Chapter 6	Data Types
Oct 28	Week #9	Chapter 7	Expressions and Assignment Statements
Nov 4	Week #10	Chapter 16	Logic Programming Languages and Prolog
Nov 11	Week #11	Sections 8.1 - 8.4, 8.6	Flow Control and Statement-Level Structures
Nov 18	Week #12	Sections 9.1 - 9.10	Subprograms and Parameters
Nov 25	Week #13	Chapter 10 (Section 10.6 optional)	Implementing Subprograms
Dec 2	Week #14	Wrap-Up and Review	
Dec 9		**Final Exam Week**	

This course will follow three general tracks:

- Implementation of Programming Languages
- Programming Language Categories
- Description and Analysis of Syntax and Semantics

The **Programming Language Categories** track will run concurrently with the other two tracks as you learn new programming languages and styles of programming.

Description and Analysis of Syntax and Semantics

Introduction and History of Programming Languages

- Chapter 1 : Preliminaries
- Chapter 2 : Evolution of the Major Programming Languages
- Language Translation and Virtual Machines

Describing Syntax and Semantics

- Chapter 3 : Describing Syntax and Semantics

Lexical and Syntax Analysis

- Chapter 4 : Lexical and Syntax Analysis

Implementation Of Programming Languages

Declarations and Types of Data

- Chapter 5 : Names, Bindings, Type Checking, and Scopes
- Chapter 6 : Data Types

Expression, Assignments & Control

- Chapter 7 : Expressions and Assignment Statements
- Chapter 8 : Statement-Level Control Structures

Subprograms and Functions

- Chapter 9 : Subprograms
- Chapter 10 : Implementing Subprograms

Advanced Language Concepts (if time permits)

- Chapter 13 : Concurrency
- Chapter 14 : Exception Handling and Event Handling

Programming Language Categories

Functional Programming Languages

- Chapter 15 : Functional Programming Languages
- Handouts on Scheme and Racket

Logic Programming Languages

- Chapter 16 : Logic Languages
- Handouts on Prolog

Object Oriented Programming Languages

- Chapter 11 : Abstract Data Types and Encapsulation
- Chapter 12 : Support for Object Oriented Programming

Some of the specific concepts we will cover are:

- **Overview of programming languages**: History of programming languages; brief survey of programming paradigms; the role of language translation in the programming process
- **Fundamental issues in language design**: General principles of language design; design goals; typing regimes; data structure models; control structure models; abstraction mechanisms
- **Virtual machines**: The concept of a virtual machine; hierarchy of virtual machines; intermediate languages
- **Introduction to language translation**: Comparison of interpreters and compilers; language translation phases; machine-dependent and machine-independent aspects of translation; language translation as a software engineering activity
- **Lexical analysis**: Application of regular expressions in lexical scanners; hand-coded vs. automatically-generated scanners; formal definition of tokens; implementation of finite-state automata
- **Syntactic analysis**: Formal definition of grammars; BNF and EBNF; bottom-up vs. top-down parsing; tabular vs. recursive-descent parsers; error handling; automatic generation of tabular parsers; symbol table management; the use of tools in support of the translation process
- **Models of execution control**: Order of evaluation of subexpressions; exceptions and exception handling; runtime systems
- **Declaration, modularity, and storage management**: Declaration models; parameterization mechanisms; type parameterization; mechanisms for sharing and restricting visibility of declarations; garbage collection
- **Type systems**: Data type as set of values with set of operations; data types; type-checking models; semantic models of user-defined types; parametric polymorphism; subtype polymorphism; type-checking algorithms
- **Interpretation**: Iterative vs. recursive interpretation; iterative interpretation of intermediate code; recursive interpretation of a parse tree
- **Code generation**: Intermediate and object code; intermediate representations; implementation of code generators; code generation by tree walking; context-sensitive translation; register use

Academic Integrity

All work is to be the result of your own individual efforts unless explicitly stated otherwise. Plagiarism, unauthorized cooperation or any form of cheating will be brought to the attention of the Dean for disciplinary action. If you have any uncertainties in this area, see the appropriate sections (8.27) of the University Manual.

You may discuss homework in a general way with other students, but you may not consult any one else's written work. Sharing of code on individual programming assignments is a form of academic dishonesty. Any similarity in form or notation between submissions with different authors will be regarded as evidence of academic dishonesty -- so protect your work.

Software piracy will be dealt with exactly like stealing of university or departmental property. Any abuse of computer or software equipment will subject to disciplinary action.

You should use your own solutions to any of the programing assignments. If you use excerpts of code from the textbook or an Open Source project, always cite your source and check the license or copyright agreement to ensure how you use the code is acceptable. In most cases, you can freely use Open Source software as long as the appropriate copyright notice is maintained in your derivative work.

Remember - cite the source of your source if it was not your creation!

Another common area of plagiarism is in homework assignments. In this internet age, it is quick and easy to do a web search on the text of a homework question and find web pages where students and even instructors, have posted solutions. To copy and paste these solutions into your assignment is plagiarism. As a warning, this type of plagiarism is very easy to discover because I can easily perform the same web search.

Here are some examples of plagiarism using source code:

Original Source (computer program):

```
quicksort (int a[ ], int 1, int r) {
     int v, i, j, t;
          if (r > 1) {
          v = a [r]; i = 1-1; j = r;
          for (;;) {
               while (a [++i] < v) ;
               while (a [-j] > v) ;
               if (i >= j) break;
               t= a[ i ];
               a[ i ] = a [r];
               a [ r ] = t;
          }
          t = a [ i ];
          a [ i ] = a [ r ]; a [ r ] = t
          quicksort (a, l, i-1);
          quicksort (a, i+1, r);
     }
}
```

From: Robert Sedgewick: The above program appears on page 118 of the textbook *Algorithms in C* (Addison Wesley, New York, 1990)

Computer Program Example 1:

```
mysort (int data[ ], int x, int y) {
     int pivot;
     int i, j;
     int temp;
     if (y > x) {
          pivot = data [ y ]; i = x-1; j = r;
          while (1) {
               while (data [ + + i ] < pivot) ;
               while (data [ - j ] > pivot) ;
               if ( i >= j) break;
               temp = data [ i ];
               data [ i ] = data [ y ];
               data [ y ] = temp;
          }
          temp = data [ i ];
          data [ i ] = data [ y ]; data [ y ] = temp;
          mysort (data, x, i-1);
          mysort (data, i+1, y);
     }
}
```

This example is plagiarism because the student has borrowed the structure of the original program exactly, while changing only a few details that do not affect the meaning of the program. Though the program looks different to the untrained eye, it has exactly the same meaning as the original program. The student has made the following changes:

- Changed the names of the variables: **a**, **l**, **r**, **v**, and **t** are changed to **data**, **x**, **y**, **pivot**, and **temp** (respectively);
- Replaced the construct "**for (; ;)**" with the equivalent construct "**while (1)**"
- Changed the name of the procedure from "**quicksort**" to "**mysort**"
- Changed the indentation and the division of program elements between lines.

The student's action can be compared to copying a block of text from a reference book with an occasional change in wording.

Computer Program Example 2:

```
#define Swap (A,B) { temp=(A); (A)=(B); (B)=(A); }
void mysort (const int * data, int x, int y) {
     int temp;
     while (y > x) {
          int pivot = data [ y ];
          int i = x-1;
          int j = r;
          while (1) {
               while (data [ + + i ] < pivot) { /*do nothing*/ }
               while (data [ - j ] > pivot) { /*do nothing*/ }
               if ( i >= j) break;
               swap (data [ i ], data [ y ];
          }
          swap (data [ i ], data [ j ];
          mysort (data, x, i-1);
          x = i+1;
     }
}
```

This example is also plagiarism. The student has made more changes to the program than in the first example, and some of this student's changes are even improvements to the program. Nevertheless, this student's program is clearly derived from the program in the textbook.

The student's action in this case can be compared to paraphrasing a passage from a reference book.