CALCULUS I:

COURSE DESCRIPTION:

Prerequisites: MAT 172; minimum grade of “C”

Corequisites: None

This course is designed to develop the topics of differential and integral calculus. Emphasis is placed on limits, continuity, derivatives and integrals of algebraic and transcendental functions of one variable. Upon completion, students should be able to select and use appropriate models and techniques for finding solutions to derivative-related problems with and without technology. Students may not receive credit for both MAT 263 and MAT 271. Course Hours per Week: Class, 3. Lab, 2. Semester Hours Credit, 4.

This is a Universal General Education Transfer Component (UGETC) course

LEARNING OUTCOMES:

1. Apply the definition of limit to evaluate limits by multiple methods and use it to derive the definition and rules for differentiation and integration.

2. Use derivatives to analyze and graph algebraic and transcendental functions.

3. Select and apply appropriate models and differentiation techniques to solve problems involving algebraic and transcendental functions; these problems will include but are not limited to applications involving optimization and related rates.

4. Apply the definition of indefinite integral to solve basic differential equations.

5. Apply the definition of definite integral to evaluate basic integrals.

6. Use the fundamental theorem of calculus to evaluate integrals involving algebraic and transcendental functions.

OUTLINE OF INSTRUCTION:

I. Limits and Derivatives

A. The Tangent and Velocity Problems

B. The Limit of a Function

C. Calculating Limits Using the Limit Laws

D. The Definition of a Limit

E. Continuity

F. Limits at Infinity; Horizontal Asymptotes

G. Derivatives and Rates of Change

H. The Derivative as a Function

II. Differentiation Rules

A. Derivatives of Polynomials and Exponential Functions

B. The Product and Quotient Rules

C. Derivatives of Trigonometric Functions

D. The Chain Rule

E. Implicit Differentiation

F. Derivatives of Logarithmic Functions

G. Rates of Change in the Natural and Social Sciences

H. Hyperbolic Functions

III. Applications of Differentiation

A. Related Rates

B. Linear Approximations and Differentials

C. Maximum and Minimum Values

D. The Mean Value Theorem

E. How Derivatives Affect the Shape of a Graph

F. Curve Sketching

G. Optimization Problems

IV. Integrals

A. Antiderivatives

B. Areas and Distances

C. The Definite Integral

D. The Fundamental Theorem of Calculus

E. Logarithm Defined as an Integral

F. Indefinite Integrals and the Net Change Theorem

G. The Substitution Rule

REQUIRED TEXTBOOK AND MATERIALS:

Stewart, James. Calculus: Early Transcendentals. 8th ed. Brooks/Cole, Cengage Learning 2012.

TI-83/84 Graphing Calculator

Foundations of Mathematics:

Course Description:

This course is designed to meet the needs of the student who has an average

background in basic mathematics, yet presents a reasonable level of abstraction for

Algebraic relations, functions, and mathematical sentences, from simple expressions

to linear equations, to general equations, involving absolute value, exponents, and

radicals. Students are admitted to this class by teacher recommendation only.

Prerequisite & Course Information:

Geometry, teacher recommendation

1 year course, open to grade 11

Course Outline Topics:

Arithmetic computations involving whole numbers, fractions, and percents.

Occupational word problems will be stressed.

*Topics will be modified when necessary

Grade Determination:

40% Test

40% Homework

20% Quiz

Expectations:

Bring your book, pencil, paper, and laptop to class.

Expect to have a weekly quiz.

Tests will be given at the end of each chapter.

Finals must be taken.

FOUNDATIONS of COMPUTER PROGRAMMING

Course Aims

• Introducing the main cultural and methodological foundations of computer programming.

• Understanding the nature, potentials and limits of programming.

• Developing basic problem solving and operational skills for programming in-the-small.

• Learning how to apply basics concepts in the functional, imperative and object-oriented paradigms in

order to solve simple problems.

• Understanding the logical properties characterizing the correct behaviour of a program.

• Understanding some basic concepts of data abstraction and object-oriented programming.

Course Organization

The course focus is on the main forms of abstraction to cope with problem complexity: procedural abstraction,

data abstraction, and abstraction of state. The procedural and data abstraction are introduced through suitable

examples from a functional perspective; the abstraction of state from an imperative and object-oriented

perspective. The laboratory sessions are aimed at developing and experimenting with programs that apply the

techniques presented in the theoretical lessons.

The course is based on a functional-first approach (IEEE-CS/ACM Computing Curricula 2001).

Prerequisites:

High school mathematics.

Course Syllabus

Part I - Procedural abstraction (Scheme)

Numerical and non-numerical expressions. Functional approach: procedural abstraction. Simple- (if) and

multiple-choice (cond) constructs. Numerical, boolean, character and string values. Recursive procedures. Wellfounded

recursive definitions. Evaluation model via substitution and reduction. Let construct. General and tail

recursion. Tree recursion and computational complexity. Correctness of recursive programs: proof by induction.

Higher order procedures.

Part II - Data abstraction (Scheme)

Simple data abstractions. Examples of linear and tree data structures: traversal and search algorithms,

applications. Algorithms and computational costs. Different implementations of abstract data types. Data

structures from the user’s (protocol/interface) vs. the implementor’s viewpoint (behavior).

Part III - Abstraction of state (Scheme and Java)

Concept of state and imperative paradigm. Basic statements and constructs of the Java language. Arrays and

array operations. Data structures and imperative approach. Top-down (memoization) and bottom-up dynamic

programming. Functional vs. imperative approach. State abstraction: concepts of class, object, constructor and

method. Encapsulation and information-hiding in the object-oriented approach. Computational costs. Examples

of implementation of dynamic data structures in Java. Verification of correctness: Assertions, loop invariants

and termination functions.

Recurrent Concepts: Procedural abstraction; computational model; recursive approach; tail recursion; tree

recursion; imperative approach; iteration; algorithm; computational complexity; preconditions and

postconditions; invariant; termination; data abstraction; state abstraction; object-oriented approach; protocol;

object; data hiding; modular organization.

Laboratory

The laboratory sessions are about design, development and experimentation of small-scale programs; they are

meant to stimulate students' organization skills as well as their ability to work autonomously.

FOUNDATIONS of COMPUTER PROGRAMMING

(teacher: Claudio Mirolo)

Textbooks

• Max Hailperin, Barbara Kaiser, Karl Knight

Concrete Abstractions: An Introduction to Computer Science Using Scheme

Brooks/Cole Publishing Company, 1999 (ISBN: 0-534-95211-9)

• Robert Sedgewick, Kevin Wayne

Introduction to Programming in Java

Addison-Wesley, 2007 (ISBN: 0-321-49805-4)

Exams

Organization of the exams:

• Two written tests, scheduled at the end of each semester;

• A few simple laboratory assignments;

• Oral discussion.

Course Title: Calculus II:

Credits: 4 Semester Hours

Prerequisites: MAT 181: Calculus I or appropriate placement.

Objectives:

After successful completion of this course a students will be able to demonstrate:

🕐 Basic knowledge of the fundamental concepts behind definite and indefinite

integration, i.e. Riemann Sums and the Fundamental Theorem of Calculus.

🕐 Procedural facility with the rules of integral calculus and with techniques for

anti-differentiation.

🕐 Basic knowledge of numerical sequences and series including tests for

convergence and methods of approximation of sums.

🕐 Basic knowledge of power and Taylor series including test for convergence and methods of approximation of sums.

 An ability to use calculus to solve some basic applied problems.

 An ability to use technological tools to represent some fundamental concepts
of calculus and to solve basic problems of application.

Course Description:

Calculus II will introduce students to a variety of new techniques of integration, to some applications of integration, and to sequences and series. Students will be expected both to become proficient with basic skills and to demonstrate an understanding of the underlying principles of the subject. Students should expect to make appropriate use of technology in this course. Knowledge of Calculus I will be assumed, in particular knowledge of the rules and concepts behind differentiation and basic integration. Prerequisite: MAT 181: Calculus I or appropriate placement.

Course Outline:

I. Prerequisite Material (not covered or only briefly reviewed)

a. Knowledge of functions: algebraic, transcendental, explicit, implicit,
and parametric.

b. An understanding of the concept of limits and continuity.

c. Facility with the rules for differentiation.

d. Definition of the definite integral as a limit of Riemann sums.

e. Facility with basic rules for anti-differentiation.

II. Requisite Material (core subjects necessary to the course)

a. Integration

i. Definite integral as a limit of Riemann Sums

ii. Fundamental Theorem of Calculus

iii. Techniques of Integration

1. Integration by substitution

2. Integration by parts

iv. Applications of Integration

1. Areas between curves

2. Volumes by slicing and revolution

v. Improper Integrals

b. Sequences and series

i. Basic introduction to sequences and the meaning of their

convergence

ii. Series

1. Convergence in terms of sequences of partial sums

2. Geometric series

3. Convergence tests

4. Alternating series

c. Power and Taylor series

i. Center and radius of convergence

ii. Functions as infinite series

iii. Approximating functions by Taylor Polynomials

III. Additional Material (topics covered at the discretion of the instructor)

a. Integration

i. Techniques of Integration

1. Use of tables of integration

2. Integration by partial fraction decomposition

3. Integration by trigonometric substitution

4. Numerical approximations of the definite integral

ii. Applications of Integration

1. Arclength

2. Work and center of mass

3. Probability

4. Economics

b. Differential Equations

i. Slope fields

ii. Euler’s method

iii. Separation of variables

iv. General applications

c. Series

i. Introduction to Fourier Series

ii. Errors in series approximations

Linear Algebra:

Course Description

Foundations to Frontiers (LAFF) is packed full of challenging, rewarding material that is essential for mathematicians, engineers, scientists, and anyone working with large datasets. Students appreciate our unique approach to teaching linear algebra because:

It’s visual.

It connects hand calculations, mathematical abstractions, and computer programming.

It illustrates the development of mathematical theory.

It’s applicable.

In this course, you will learn all the standard topics that are taught in typical undergraduate linear algebra courses all over the world, but using our unique method, you'll also get more! LAFF was developed following the syllabus of an introductory linear algebra course at The University of Texas at

Austin taught by Professor Robert van de Geijn, an expert on high performance linear algebra libraries. Through short videos, exercises, visualizations, and programming assignments, you will study Vector and Matrix Operations, Linear Transformations, Solving Systems of Equations, Vector Spaces, Linear Least-Squares, and Eigenvalues and Eigenvectors. In addition, you will get a glimpse of cutting edge research on the development of linear algebra libraries, which are used throughout computational science.

MATLAB licenses will be made available to the participants free of charge for the duration of the course.

We invite you to LAFF with us!

See more about Linear Algebra - Foundations to Frontiers

What you'll learn

Connections between linear transformations, matrices, and systems of linear equations

Partitioned matrices and characteristics of special matrices

Algorithms for matrix computations and solving systems of equations

Vector spaces, subspaces, and characterizations of linear independence

Orthogonality, linear least-squares, eigenvalues and eigenvectors

Hide Course Syllabus

Course Syllabus

Skip Syllabus DescriptionWeek 0 Get ready, set, go!

Week 1 Vectors in Linear Algebra

Week 2 Linear Transformations and Matrices

Week 3 Matrix-Vector Operations

Week 4 From Matrix-Vector Multiplication to Matrix-Matrix Multiplication

Exam 1

Week 5 Matrix-Matrix Multiplication

Week 6 Gaussian Elimination

Week 7 More Gaussian Elimination and Matrix Inversion

Week 8 More on Matrix Inversion

Exam 2

Week 9 Vector Spaces

Week 10 Vector Spaces, Orthogonality, and Linear Least Squares

Week 11 Orthogonal Projection and Low Rank Approximation

Week 12 Eigenvalues and Eigenvectors

Final

Advanced Programming:

Course Contents3 Units

The purpose of the course is to study the fundamental concepts and techniques necessary to write high-quality programs, including basic concepts of object-oriented programming, modular design, exception handling, and class libraries. Some advanced topics such as reflection, distributed programming, multi-threading, and GUI libraries are also covered. All of the mentioned concepts and techniques are studied using the Java language. It is important to note that this course is not a Java training course. The emphasis is on the concepts and techniques rather than the language itself.

Prerequisites

It is assumed that students are familiar with the following:

At least one structured programming language such as Pascal or C.

Basic data models: Linked Lists, Sets, Trees, and Graphs.

Basic concepts of operating systems.

Textbooks:

Bruce Eckel, Thinking in Java, 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2000.

Harvey M. Deitel, and Paul J. Deitel, Java How to Program, 3rd ed., Prentice-Hall, Upper-saddle River, NJ, 2001

Foundation of Analysis:

The course provides an opportunity for the development of theorem-proving skills in the field of mathematical analysis. Expansion of a knowledge base comes as a by-product of energy expended in theorem proving and subsequent exposition. Analysis topics included are: sets, functions, the real numbers, cardinality, induction, decimal representations of real numbers, Euclidean spaces, abstract vector spaces, and metric spaces. This is a communication-intensive course.

Prerequisites/Corequisites: Prerequisite: Mathematics major, Corequisite: MATH 2010 or MATH 2011 or permission of instructor.

When Offered: Fall and spring terms annually.

Credit Hours: 4

Foundations of Combinatorics:

Summary: This three semester topics course on combinatorics includes Enumeration,
Graph theory, and Algebraic Combinatorics. Combinatorics has connections to
all areas of mathematics and many other sciences including biology, physics, computer
science, and chemistry. We have chosen core areas of study which should be relevant
to a wide audience. The main distinction between this course and its undergraduate
counterpart will be the pace and depth of coverage. In addition we will assume students
have a basic knowledge of linear and abstract algebra. We will include many
unsolved problems and directions for future research. The outline for each quarter is

the following:

(1) Enumeration: Every discrete process leads to questions of existence, enumeration and optimization. This is the foundation of combinatorics. In this quarter we will present the basic combinatorial objects and methods for counting various arrangements of these objects.

(a) Basic counting methods.

(b) Sets, multisets, permutations, and graphs.

(c) Inclusion-exclusion.

(d) Recurrence relations and integer sequences.

(e) Generating functions.

(f) Partially ordered sets.

(g) Complexity Theory

(2) Random Graphs, Random Groups:

Fifty years ago Erd¨os and Renyi began the study of random graphs. They answered questions such as how many randomly chosen edges does a graph on 100 vertices need to have before it is connected. The field they developed was not only able to answer probabilistic questions, but also questions about the existence of certain types of graphs. More recently Gromov started asking similar questions about what does a random group looks like. He conjectured (and Ollivier proved) that a random group with high probability is either hyperbolic or trivial.

In this class we will look at these two closely related fields. We will touch on many branches of mathematics including graph theory, probability, topology and geometric group theory. While all of these are useful, none of them are prerequisites.

(3) Algebraic Combinatorics: The focus of this class will be on combinatorial Hopf algebras and diagram algebras. Diagram algebras/groups generalize the group algebra of the symmetric group where multiplication can be defined in term of concatenating string diagrams. Important examples include braid groups, the

1

2

Braurer algebra, the Temperley-Lieb algebra, and topological quantum field theory on manifolds which appear in connection with knot theory. Combinatorial Hopf algebras have bases indexed by familiar objection in enumerative combinatorics. The primary example is the symmetric functions in a polynomial ring with n variables. This course will survey recent work on symmetric functions, quasisymmetric functions, noncommuting symmetric functions as Hopf algebras and related representation theory including 0-Hecke algebra representations.

There is no text but resources include Vic Reiner's notes on ”Hopf Algebras In Combinatorics”, Federico Ardila's lectures on ”Hopf Algebras” and the book “Coxeter Groups and Hopf Algebras” by Marcelo Aguiar and Swapneel Mahajan.

Course requirements: a strong background in algebra on par with our Math 504/5/6 and basic enumerative combinatorics on par with 461/462. No prior knowledge of Hopf algebras will be assumed.

Tentative Textbook Selection: Enumerative Combinatorics; Volume 1 by Richard Stanley, Cambridge Studies in Advanced Mathematics, 49, 1997. (First quarter only).

Data Structures and Algorithms:

Catalog Description: Fundamental algorithms and data structures for implementation. Techniques for solving problems by programming. Linked lists, stacks, queues, directed graphs. Trees: representations, traversals. Searching (hashing, binary search trees, multiway trees). Garbage collection, memory management. Internal and external sorting. Intended for non-majors. Not open for credit to students who have completed CSE 332. Prerequisite: CSE 143.

Prerequisites: CSE 143

Credits: 4.0

CALCULUS III:

COURSE DESCRIPTION:

Prerequisites: MAT 272; minimum grade of "C"

Corequisites: None

This course is designed to develop the topics of multivariate calculus. Emphasis is placed on multivariate functions, partial derivatives, multiple integration, solid analytical geometry, vector valued functions, and line and surface integrals. Upon completion, students should be able to select and use appropriate models and techniques for finding the solution to multivariate-related problems with and without technology. This course has been approved to satisfy the Comprehensive Articulation Agreement for the general education core requirement in natural sciences/mathematics. Course Hours Per Week: Class, 3. Lab, 2. Semester Hours Credit, 4.

LEARNING OUTCOMES:

1. Perform operations with vectors in two and three dimensional space and apply to analytic geometry

2. Differentiate and integrate vector-valued functions and apply calculus to motion problems in two and three dimensional space

3. Determine the limits, derivatives, gradients, and integrals of multivariate functions

4. Solve problems in multiple integration using rectangular, cylindrical, and spherical coordinate

systems

5. Select and apply appropriate models and techniques to define and evaluate line and surface integrals; these techniques will include but are not limited to Green's, Divergence, and Stoke's theorems

6. Demonstrate proficiency in using CAS technology to analyze, solve and interpret the various applications

OUTLINE OF INSTRUCTION:

I. Vectors and the Geometry of Space

A. Three-Dimensional Coordinate Systems

B. Vectors

C. The Dot Product

D.The Cross Product

E. Equations of Lines and Planes

F. Cylinders and Quadric Surfaces

MAT 273: July 2015

II. Vector Functions

A. Vector Functions and Space Curves

B. Derivatives and Integrals of Vector Functions

C. Arc Lengths and Curvature

D. Motion in Space: Velocity and Acceleration

III. Partial Derivatives

A. Functions of Several Variables

B. Limits and Continuity

C. Partial Derivatives

D.Tangent Planes and Linear Approximations

E. The Chain Rule

F. Directional Derivatives and the Gradient Vector

G. Maximum and Minimum Values

H.Lagrange Multipliers

IV. Multiple Integrals

A. Double Integrals Over Rectangles

B. Iterated Integrals

C. Double Integrals Over General Regions

D. Double Integrals in Polar Coordinates

E. Applications of Double Integrals

F. Triple Integrals

G.Triple Integrals in Cylindrical Coordinates

H.Triple Integrals in Spherical Coordinates

I. Change of Variables in Multiple Integrals

V. Vector Calculus

A. Vector Fields

B. Line Integrals

C. The Fundamental Theorem of Line Integrals

D. Green's Theorem

E. Curl and Divergence

F. Parametric Surfaces and Their Areas

G. Surface Integrals

H. Stokes Theorem

I. The Divergence Theorem

REQUIRED TEXTBOOK AND MATERIALS:

Stewart, James. Calculus Early Transcendentals. 8

th ed. Brooks/Cole, 2012

TI-83/84 graphing calculator

Principles of Operating Systems :

Course description: Introduction to the fundamental principles of operating system design. The concepts and algorithms covered in the course are based on those used in both commercial and open-source operating systems. We present these concepts and algorithms in a general setting that is not tied to one particular operating system. However, we present a large number of examples that pertain to the most popular and the most innovative operating systems, including Linux, Microsoft Windows, Apple Mac, Solaris, Android and iOS. We also assign some simple coding labs to help students understand important knowledge and representative algorithms used in operating systems, such as CPU scheduling, synchronization and virtual memory.

principles of computer systems:

Summary

This advanced graduate course focuses on key design principles underlying successful computer and communication systems, and teaches how to solve real problems using ideas, techniques, and algorithms from operating systems, networks, databases, programming languages, and computer architecture.

Content

A modern computer system spans many layers: applications, libraries, operating systems, networks, and hardware devices. Building a good system entails making the right trade-offs (e.g., between performance, durability, and correctness) and understanding emergent behaviors - the difference between great system designers and average ones is that the really good ones make these trade-offs in a principled fashion, not by trial-and-error.

In this course we develop such a principled framework for system design, covering the following topics:

Modularity, Abstraction, and Layering

Indirection and Naming

Locality

End-to-end / State partitioning

Virtualization

Atomicity and Consistency

Redundancy and Availability

Interpretation, Simulation, Declarativity

Laziness vs. Speculation

CAP Theorem, DQ Principle, Harvest/Yield

Least Privilege, Minimum TCB

Learning Prerequisites

Required courses

Principles of Computer Systems (POCS) is targeted at students who wish to acquire a deep understanding of computer system design or pursue research in systems. It is an intellectually challenging, fast paced course, in which mere survival requires a solid background in operating systems, databases, networking, programming languages, and computer architecture. The basic courses on these topics teach how the elemental parts of modern systems work - POCS picks up where the basic courses leave off and focuses on how the pieces come together to form useful, efficient systems. To do well in POCS, a student must master the material of the following courses:

COM-208 Computer networks

CS-208 Computer architecture

CS-210 Functional programming

CS-305 Software engineering

CS-322 Introduction to database systems

CS-323 Operating systems

Recommended courses

The following EPFL courses cover material that significantly help students' understanding of POCS concepts; however, these courses are not strictly required:

CS-320: Computer language processing

CS-470: Advanced computer architecture

CS-422: Database systems

COM-407: TCP/IP networking

Learning Outcomes

By the end of the course, the student must be able to:

Design computer and communication systems that work well

Make design trade-offs (e.g., performance vs. correctness, latency vs. availability)

Anticipate emergent system behaviors (e.g., failure cascades, security vulnerabilities)

Integrate multiple techniques, ideas, and algorithms from different fields of computing/communication into a working system

Teaching methods

Online video lectures

Ex cathedra

Small-group discussions and exercises

foundation of computing:

This course presents some formal notations that are commonly used for the description of computation and of computing systems, for the specification of software and for mathematically rigorous arguments about program properties. The following areas of study constitute the backbone of the course. Predicate calculus and natural deduction, inductive definitions of data types as a basis for recursive functions and structural induction, formal language theory (particularly regular expressions, finite state machines and context free grammars), and specification languages.

Learning Outcomes

Apply the concepts of standard mathematical logic to produce proofs or refutations of well-formed propositions or arguments phrased in English or in a variety of formal notations (first order logic, discrete mathematics or Hoare Logic).

Given a description of a regular language, either in English, as a regular expression or as a grammar, generate a finite state automaton that recognizes that language. Similarly, given a deterministic or nondeterministic automaton, give a description of the language which it accepts.

Given an inductive definition of a simple data structure, write a recursive definition of a given simple operation on data of that type. Given some such recursively defined operations, prove simple properties of these functions using the appropriate structural induction principle.

Prove simple programs correct using Hoare Logic.

Design a Turing Machine which will accomplish simple tasks.

Database :

Concepts :This course introduces database design and creation using a DBMS product. Emphasis is on data dictionaries, normalization, data integrity, data modeling, and creation of simple tables, queries, reports, and forms. Upon completion, students should be able to design and implement normalized database structures by creating simple database tables, queries, reports, and forms.

Course Hours Per Week: Class, 2; Lab, 3

Semester Hours Credit: 3

Prerequisite: None

Corequisite: None

design and analysis of algorithms :

Prerequisites:This course is the header course for the Theory of Computation concentration. You are expected, and strongly encouraged, to have taken:

6.006 Introduction to Algorithms

6.042J / 18.062J Mathematics for Computer Science

Petitions for waivers will be considered by the course staff. Students will be responsible for material covered in prerequisites.

Course Description:This course assumes that students know how to analyze simple algorithms and data structures from having taken 6.006. It introduces students to the design of computer algorithms, as well as analysis of sophisticated algorithms.

Course Objectives

Upon completion of this course, students will be able to do the following:

Analyze the asymptotic performance of algorithms.

Write rigorous correctness proofs for algorithms.

Demonstrate a familiarity with major algorithms and data structures.

Apply important algorithmic design paradigms and methods of analysis.

Synthesize efficient algorithms in common engineering design situations.

Course Outcomes

Students who complete the course will have demonstrated the ability to do the following:

Argue the correctness of algorithms using inductive proofs and invariants.

Analyze worst-case running times of algorithms using asymptotic analysis.

Describe the divide-and-conquer paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize divide-and-conquer algorithms. Derive and solve recurrences describing the performance of divide-and-conquer algorithms.

Describe the dynamic-programming paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize dynamic-programming algorithms, and analyze them.

Describe the greedy paradigm and explain when an algorithmic design situation calls for it. Recite algorithms that employ this paradigm. Synthesize greedy algorithms, and analyze them.

Explain the major graph algorithms and their analyses. Employ graphs to model engineering problems, when appropriate. Synthesize new graph algorithms and algorithms that employ graph computations as key components, and analyze them.

Explain the different ways to analyze randomized algorithms (expected running time, probability of error). Recite algorithms that employ randomization. Explain the difference between a randomized algorithm and an algorithm with probabilistic inputs.

Analyze randomized algorithms. Employ indicator random variables and linearity of expectation to perform the analyses. Recite analyses of algorithms that employ this method of analysis.

Explain what amortized running time is and what it is good for. Describe the different methods of amortized analysis (aggregate analysis, accounting, potential method). Perform amortized analysis.

Explain what competitive analysis is and to which situations it applies. Perform competitive analysis.Compare between different data structures. Pick an appropriate data structure for a design situation.Explain what an approximation algorithm is, and the benefit of using approximation algorithms. Be familiar with some approximation algorithms, including algorithms that are PTAS or FPTAS. Analyze the approximation factor of an algorithm.

Textbook:The primary written reference for the course is:

Buy at MIT Press Buy at Amazon Cormen, Thomas, Charles Leiserson, et al. Introduction to Algorithms. 3rd ed. MIT Press, 2009. ISBN: 9780262033848. [Preview with Google Books]

In previous semesters the course has used the first or second edition of this text. We will be using material and exercise numbering from the third edition, making earlier editions unsuitable as substitutes.

compiler:

Course Description:This module introduces topics include compiler design, lexical analysis, parsing, symbol tables,

declaration and storage management, code generation, and optimization techniques.

Course Objectives:

The aim of this module is to show how to apply the theory of language translation introduced in the prerequisite courses to build compilers and interpreters. It covers the building of translators both from scratch and using compiler generators. In the process, the module also identifies and explores the main issues of the design of translators.

The construction of a compiler/interpreter for a small language is a necessary component of this module, so students can obtain the necessary skills.

Course Components:

• Introduction to Compilers

• Lexical Analysis

• Syntax Analysis

• Parsers Implementation

• Semantic Analysis

• Intermediate Representation, code generation

• Code generation and Code optimization

• Error Detection and Recovery

• Error Repair, Compiler Implementation

Page 2 of 5

• Compiler design options and examples: C Compilers

• C++, Java, and YACC Compilers

Text book:

Title: Compilers Principles, Techniques and Tools

Author(s)/Editor(s): Alfred V. Aho, Ravi Sethi and Jeffry D. Ulman

Publisher: Addison Wesley Longman, 1986

ISBN: 0- 201- 10088- 6

In addition to the above, the students will be provided with handouts by the lecturer.

foundation of logic and set theory:

Description:Examines the logical foundations of concepts used throughout mathematics, such as order and equivalence relations, number and continuity. The use of infinity in mathematical arguments is investigated and implicit assumptions about infinite sets are exposed. Notions of infinity are formulated precisely and it is shown how infinite sets may be counted and compared in size. It is seen that, even in something as basic as set theory, 'truth' is not absolute.

Learning outcomes:On successful completion of the course students will be able to:

1. Learn about the logical foundations of such mathematical concepts as number, continuity and set

2. Gain an appreciation of the usefulness and limitations of the development of theories from axioms

3. Understand the concept of infinity and its role in mathematics.

Content:The need for a rigorous treatment of the infinite in mathematics

The Zermelo-Fraenkel Axioms

Order

Ordinal and cardinal numbers

Transfinite induction

The Axiom of Choice

The Continuum Hypothesis.

Assumed knowledge

MATH2320 or MATH2330

Assessment items:Written Assignment: Assignments

In Term Test: Mid Semester Test

Formal Examination: Examination

Linear Optimization:

Course Description

Linear optimization (or linear programming, LP) is the fundamental branch of optimization, with applications to many areas including life sciences, computer science, defense, finance, telecommunications, transportation, etc. Other types of optimization typically use LP as the underlying model. This course will provide an integrated view of the theory, solution techniques, and applications of linear optimization. There will be a fair bit of emphasis on theorems and their proofs. The treatment of most topics will begin with a geometric point of view, followed by the development of the solution techniques (algorithms), which are described using linear algebra. A background in linear algebra and multivariate calculus is assumed. Topics covered include linear programming formulations, geometry of linear programming, the simplex method, duality, sensitivity analysis, interior point methods, and integer programming basics. Apart from problems involving proofs, the student will use Octave (or Matlab) or another programming language (e.g., Python) for implementing some of the computations and algorithms. A state-of-the-art modeling software such as AMPL will also be introduced for solving problems modeling real life situations.

Foundations of Algebra:

Foundations of Algebra is a first year high school mathematics course option for students who have completed

mathematics in grades 6 – 8 yet will need substantial support to bolster success in high school mathematics. The

course is aimed at students who have reported low standardized test performance in prior grades and/or have

demonstrated significant difficulties in previous mathematics classes.

Foundations of Algebra will provide many opportunities to revisit and expand the understanding of

foundational algebra concepts, will employ diagnostic means to offer focused interventions, and will incorporate

varied instructional strategies to prepare students for required high school mathematics courses. The course will

emphasize both algebra and numeracy in a variety of contexts including number sense, proportional reasoning,

quantitative reasoning with functions, and solving equations and inequalities.

Instruction and assessment should include the appropriate use of manipulatives and technology. Mathematics

concepts should be represented in multiple ways, such as concrete/pictorial, verbal/written, numeric/data-based,

graphical, and symbolic. Concepts should be introduced and used, where appropriate, in the context of realistic

experiences.

The Standards for Mathematical Practice will provide the foundation for instruction and assessment. The content

standards are an amalgamation of mathematical standards addressed in grades 3 through high school. The

standards from which the course standards are drawn are identified for reference.

Coding theory:

Course Description:This course introduces the theory of error-correcting codes to computer scientists. This theory, dating back to the works of Shannon and Hamming from the late 40's, overflows with theorems, techniques, and notions of interest to theoretical computer scientists. The course will focus on results of asymptotic and algorithmic significance. Principal topics include:

Construction and existence results for error-correcting codes.

Limitations on the combinatorial performance of error-correcting codes.

Decoding algorithms.

Applications in computer science.

Computer Logic Circuits

COURSE DESCRIPTION:This course provides the student with a foundation in the fundamentals of digital logic design and

computer logic circuits. Both combinational and sequential logic circuits are covered in this course. The

emphasis is on the use of Boolean algebra and basic logic gates to build cost effective complex logic

circuits. Topics include: Number systems, Binary arithmetic, Codes, Logic gates, Boolean algebra and

simplifications, Half adders, Full adders, Decoders, Encoders, Multiplexers, Latches, Flip-Flops,

Counters, Shift Registers, Memory circuits, and ALU (Arithmetic and Logic Unit).

COURSE OBJECTIVES:

After completing the course, the student will be able to:

1. understand number systems, codes, and binary arithmetic,

2. apply Boolean algebra to logic design,

3. produce the truth table, timing diagram and gate design of logic functions,

4. simplify logic functions for cost-effective implementation,

5. build and trace the logic of circuits composed of simple gates,

6. describe the behavior of the following circuits: Half Adder, Full Adder, Decoder, Encoder,

Multiplexer, Latches, Flip-Flops, Registers, Counters, and Memory,

7. understand the structure of an ALU, and trace the logic of simple instruction execution,

8. write simple programs in Verilog language to describe simple logic circuits.

MAJOR TOPICS:

1. Introduction to Computer Logic Circuits

2. Number Systems, Binary Arithmetic, and Codes

3. Boolean algebra and Logic Gates

4. The Karnaugh Map and Logic Simplifications

5. Combinational Logic Circuits

a. Half Adders and Full Adders

b. Comparators

c. Encoders and Decoders

d. Code Conversions

e. Multiplexers and De-multiplexers

6. Latches and Flip-Flops

7. Counters

8. Shift Registers

9. Memory Circuits

10. Arithmetic and Logic Unit

11. Hardware Description Language (Verilog)

Data Mining:

Brief Course Description:Data mining, or knowledge discovery in databases, has during the last few years emerged as one of the most exciting fields in Computer Science. Data mining aims at finding useful regularities in large data sets. Interest in the field is motivated by the growth of computerized data collections which are routinely kept by many organizations and commercial enterprises, and by the high potential value of patterns discovered in those collections. For instance, bar code readers at supermarkets produce extensive amounts of data about purchases. An analysis of this data can reveal previously unknown, yet useful information about the shopping behavior of the customers.

Data mining refers to a set of techniques that have been designed to efficiently find interesting pieces of information or knowledge in large amounts of data. Association rules, for instance, are a class of patterns that tell which products tend to be purchased together. There is currently a large commercial interest in the area, both for the development of data mining software and for the offering of consulting services on data mining, with a market for the former estimated at over 5 billion U.S. dollars.

In this course we explore how this interdisciplinary field brings together techniques from databases, statistics, machine learning, and information retrieval. We will discuss the main data mining methods currently used, including data warehousing and data cleaning, clustering, classification, association rules mining, query flocks, text indexing and seaching algorithms, how search engines rank pages, and recent techniques for web mining. Designing algorithms for these tasks is difficult because the input data sets are very large, and the tasks may be very complex. One of the main focuses in the field is the integration of these algorithms with relational databases and the mining of information from semi-structured data, and we will examine the additional complications that come up in this case.

Artificial intelligence:

Course description:Artificial intelligence (AI) is a research field that studies how to realize the intelligent human behaviors on a computer. The ultimate goal of AI is to make a computer that can learn, plan, and solve problems autonomously. Although AI has been studied for more than half a century, we still cannot make a computer that is as intelligent as a human in all aspects. However, we do have many successful applications. In some cases, the computer equipped with AI technology can be even more intelligent than us. The Deep Blue system which defeated the world chess champion is a well-know example.

The main research topics in AI include: problem solving, reasoning, planning, natural language understanding, computer vision, automatic programming, machine learning, and so on. Of course, these topics are closely related with each other. For example, the knowledge acquired through learning can be used both for problem solving and for reasoning. In fact, the skill for problem solving itself should be acquired through learning. Also, methods for problem solving are useful both for reasoning and planning. Further, both natural language understanding and computer vision can be solved using methods developed in the field of pattern recognition.

In this course, we will study the most fundamental knowledge for understanding AI. We will introduce some basic search algorithms for problem solving; knowledge representation and reasoning; pattern recognition; fuzzy logic; and neural networks.

Principles of Software Design:

About this course: Solve real world problems with Java using multiple classes. Learn how to create programming solutions that scale using Java interfaces. Recognize that software engineering is more than writing code - it also involves logical thinking and design. By the end of this course you will have written a program that analyzes and sorts earthquake data, and developed a predictive text generator.

After completing this course, you will be able to:

1. Use sorting appropriately in solving problems;

2. Develop classes that implement the Comparable interface;

3. Use timing data to analyze empirical performance;

4. Break problems into multiple classes, each with their own methods;

5. Determine if a class from the Java API can be used in solving a particular problem;

6. Implement programming solutions using multiple approaches and recognize tradeoffs;

7. Use object-oriented concepts including interfaces and abstract classes when developing programs;

8. Appropriately hide implementation decisions so they are not visible in public methods; and

9. Recognize the limitations of algorithms and Java programs in solving problems.

10. Recognize standard Java classes and idioms including exception-handling, static methods, java.net, and java.io packages.