CIS 1.5 Course Objectives

By the end of this course, students should

a. Understand the concept of a program (i.e., a computer following a series of instructions)

b. Understand the concept of a variable holding a value, how a variable is declared and how it can change

c. Understand the concept of a loop – that is, a series of statements which is written once but executed repeatedly- and how to use it in a programming language

d. Be able to use a conditional statement to select a choice from two or more alternatives

e. Be able to break a large problem into smaller parts, writing each part as a module or function

f. Be able to use an array to store multiple pieces of homogeneous data, and use a structure to store multiple pieces of heterogeneous data

g. Be able to work with both character and numerical data

h. Understand the concept of an algorithm (that is, a series of steps that can be carried out in a mechanical way) and a few specific examples of algorithms (for example, finding an average, sorting, searching)

i. Understand the parts of a computer system and how they interact

j. Understand the concept of a program in a high-level language being translated by a compiler into machine language program and then executed

CIS 4.1 Course Objectives

By the end of this course, students should be able to

a. understand the difference between registers and memory, the association of registers with segments, and the implementation of segmentation for X86 based computers

b. understand the relationship of assembly language and the architecture of the machine; this includes the addressing system, how instructions and variables are stored in memory, and the fetch-and-execute cycle

c. use basic assembly language instructions

d. understand the role of interrupts in performing I/O

e. declare and use variables, and write a structured assembly language program

f. write and use functions, and understand how function calls are carried out, including passing parameters

g. understand the relationship between high level constructs such as loops, if-else, functions, arrays, and structures, macros, and their underlying representation; be able to write those constructs in assembler

h. know how to call an assembly language routine from a high level language and vice versa

i. do conversions and basic arithmetic operations in the binary, decimal and hexadecimal bases, and convert numbers to two's complement notation; use hex to calculate offsets and addresses

CIS 11 Course Objectives

By course-end the student will be able to understand and use:

a. Logical propositions (including quantifiers).

b. Simple proofs of mathematical statements (mathematical induction, indirect arguments).

c. Functional and relational properties (one-to-one, onto, reflexive, symmetric, transitive, equivalence, partial ordering), and operations (composition, transitive closure).

d. Basic matrix operations.

e. Fundamental concepts of set theory and Boolean Algebra.

f. Counting principles, countable and uncountable sets.

g. Basic probability theory and applications.

h. Recursive definitions and solutions of simple of recurrence relations.

i. Graph algorithms and their application to computer science.

j. Tree traversal algorithms.

CIS 15 Course Objectives

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

a. interpret program specifications and test their implementation,

b. write programs using a core portion of the C run-time library including: input/output, string conversion and manipulation, dynamic memory allocation, environment access;

c. program effectively with pointers, arrays, structures, and dynamically allocated memory and describe their internal representations;

d. discuss the consequences and relative merits of compile-time and run-time memory allocation;

e. employ a modular approach to program development by effective use of C's support of separate compilation including header files and external and static declarations;

f. discuss the machinery of compilation and execution, including compilation phases, object modules, link editing, and program execution;

g. implement recursive solutions to problems and demonstrate how recursion is implemented by drawing diagrams of and tracing changes in the runtime stack;

h. implement generic functions using function pointers and utilize generic functions in modular program development;

i. to describe the internal representation of C's primitive data types and strings using byte diagrams; and

j. use the programming environment offered by a Unix-like system including writing scripts that employ its utilities and writing programs that utilize its programming interface.

CIS 22 Course Objectives

By the end of the course, students should be able to:

a. Demonstrate understanding of the abstract properties of various data structures such as stacks, queues, lists, and trees.
b. Use various data structures effectively in application programs.
c. Implement various data structures in more than one manner.
d. Compare different implementations of data structures and to recognize the advantages and disadvantages of the different implementations.
e. Demonstrate understanding of and be able to program various sorting algorithms, including bubble sort, insertion sort, selection sort, heapsort and quicksort.
f. Compare the efficiency of various sorting algorithms in terms of both time and space.
g. Program multiple file programs in a manner that allows for reusability of code.
h. Trace and code recursive functions.
i. Know features of C++ and be able to program with C++ classes.
j. Demonstrate some understanding of object-oriented programming.

CIS 23 Course Objectives

By the end of the course, students should:

a. Demonstrate an understanding of the growth of functions and the hierarchy of complexity classes.
b. Demonstrate an understanding of the use of O, Ω, and Θ notation.
c. Demonstrate an understanding of the concepts of worst-case and average-case performance, upper and lower bounds.
d. Be able to analyze the complexity and efficiency of an algorithm in terms of time and space.
e. Demonstrate knowledge of algorithms for order statistics.
f. Demonstrate knowledge of several algorithms for sorting and be able to compare their complexities.
g. Demonstrate an understanding of various design techniques such as divide-and-conquer and greedy methods.
h. Demonstrate knowledge of different methods for representing a graph.
i. Be able to trace, implement, and analyze graph algorithms such as traversals, finding a minimum spanning tree and finding the shortest path.
j. Be able to demonstrate an understanding of the nature of the classes P, NP, and NP-complete, be able to define several problems that belong to these classes and be able to prove their classification.

CIS 24 Course Objectives

Upon completing the course, the student will:

a. be able to describe the salient characteristics of several language paradigms (procedural, object-oriented, imperative, declarative/logic, functional).

b. understand the concept of data binding and its effect upon the semantic level of the language.

c. be familiar with standard mechanisms of realizing language semantics at execution time.

d. understand the spectrum of source-to-executable language translation, its effect upon efficiency and expressivity the corresponding relation to data binding.

e. be able to use formal techniques (such as BNF) in the specification language syntax.

f. be able to recognize the relationship between the semantic level of the language and its expressivity, efficiency, control mechanisms, and data types.

g. be able to apply the conceptual material covered in this course (i.e. , binding times, run-time support etc.) to the analysis of specific languages.

h. be able identify the core semantics of data types and control constructs and to recognize the similarity and differences between data and control representations of various languages.

i. be able to code small programs that illustrate the core semantics of each of set of languages that represent the paradigms covered in the course.

j. be able to discuss the technological, software-engineering, and educational issues that propelled the evolution of programming languages.

CIS 25 Course Objectives

By the end of this course, students should be able to:

a. demonstrate the ability to write a large program (with subprocesses) that implements techniques taught in the course.

b. demonstrate understanding of the file system, secondary storage management and disk scheduling algorithms.

c. demonstrate understanding of the principles of multiprogramming.

d. demonstrate understanding of the methods of inter-process communication and synchronization.

e. demonstrate understanding of the issues of deadlock and its solutions.

f. demonstrate understanding of memory management and virtual memory.

g. demonstrate understanding of processes and threads.

h. demonstrate understanding of CPU scheduling and long-term scheduling.

i. demonstrate understanding of the various forms of I/O device interaction with the operating system (e.g., interrupts).

j. demonstrate understanding of the basics of computer security.

CIS 26 Course Objectives

By the end of this course, students should be able to:

a. Understand and apply the concepts of class, object, instantiation, and methods.
b. Understand and apply the concepts of data abstraction, inheritance (single and multiple), and polymorphism.
c. Write applications in an object-oriented language.
d. Understand an use run-time exceptions.
e. Understand and apply the basic concepts of file I/O.
f. Understanding and apply the concepts of multi-threading.
g. Understanding and apply the concepts of event-driven programming.
h. Evaluate the appropriateness of a specified class hierarchy for a given task.

CIS 38 Course Objectives

By course-end, students will be able to:

a. Understand the difference between deterministic and non-deterministic finite state automata

b. Design deterministic and non-deterministic finite state automata for language recognition (and minimize them)
c. Devise regular expressions for languages
d. Design push-down automata for language recognition
e. Use the pumping lemma to classify languages as not regular
f. Design simple context-free grammars
g. Design simple Turing machines
h. Understand the limits of computation via the halting problem and the Church-Turing thesis
i. Understand the basic concepts of the theory of computational complexity
j. Understand reducibility among problems and complexity classes.

CIS 60.1 Course Objectives

By the end of this course, students should:

a. Have developed skills in computer programming.

b. Have developed skills in developing algorithms and transforming algorithms
c. into a plan for solution.

d. Gain breadth in several areas in computer science.

e. Be able to identify the goals, methods, tools and outcomes of a project.

f. Be able to communicate effectively.

g. Be able to record and document the results of the project.

h. Have an understanding of the project life cycle, including how milestones are noted, monitored and revised.

CIS 88.1 Course Objectives

By the end of this course, students should:

a. Be able to apply problem-solving and programming skills.

b. Be able to communicate effectively.

c. Have acquired a fundamental understanding of how the relevant branches of computer science are used in a research project.

d. Be able to identify the goals, methods, tools and outcomes of a research project.

e. Be able to express in writing the motivation of a research project, including an overview of the background and the ongoing work with references.

f. Become an efficient user of the methods and tools used in a research project.

g. Be able to record, document, and present new findings of a research project as soon as they are obtained through observation, computation or experimentation.

h. Have gained an appreciation for ambiguity, complexity and randomness in solving problems.

i. Have created new knowledge based on the synthesis or analysis of the new findings of a research project.