

Syllabus

CSC 460: Language Translation

A. Protocol

1. Course name: Language Translation
2. Course number: CSC 460
3. Credits: Three (3). Three one hour or two one and one-half hour lectures per week.
4. Prerequisite(s): CSC 323 (CSC 270)
Upon entering the course, the student must be able to:
 - a. Discuss and illustrate the definition of data structures.
 - b. Discuss the concept of pointers.
 - c. Discuss and illustrate the design of stacks, queues, linked lists, and trees.
 - d. Write programs which require stacks, queues, linked lists, and binary trees.
 - e. Write subprograms that perform the basic stack, queue, linked list, and binary search tree operations.
 - f. Discuss the applications of data structures.
 - g. Discuss the management of computer memory using elementary abstract data structures.
 - h. Perform a binary search in a computer program.
 - i. Discuss hashing.
 - j. Write programs with good style, efficiency, and documentation.
 - k. Write large computer programs containing many subprograms
 - l. Document programs well by including a prologue to the main program and each subprogram and a liberal dispersal of comments in each module.
 - m. Discuss briefly the fetch-execution of the VAX computer.
 - n. Use VAX instructions to create selection structures.
 - o. Use VAX instructions to create looping structures.
 - p. Use the VAX Debugger with an assembly language program.
 - q. Allocate memory for integers of various sizes and perform operations on those memory locations.
 - r. Write and use macros in their programs.
 - s. Allocate arrays and manipulate them using the VAX Assembly language.
 - t. Discuss indirect addressing and implement various forms of indirect addressing, including the register-deferred, autoincrement, autodecrement, displacement, relative-deferred, and two-levels of indirection addressing modes.
 - u. Allocate character locations and manipulate character data.
 - v. Implement subprograms in the VAX Assembly language, including parameter passing and recursion.
 - w. Allocate floating-point memory locations and perform floating-point operations.
5. Date of revision: Fall 1992

B. Objectives of the Course

Upon completion of this course the student will be able to:

1. Identify the phases of a compiler.
2. Design a data structure to store tokens.
3. Discuss the creation of symbol tables.
4. Discuss error analysis of computer programs.
5. Distinguish between lexical and syntactic errors.
6. Write transition diagrams recognizing tokens.
7. Specify tokens using regular expressions.
8. Design a Lexical Analyzer using transition diagrams.
9. Use syntax definition to analyze programs and statements.
10. Use translation of postfix notation to analyze perform code generation of assignment statements.
11. Discuss top down, recursive descent, and predictive parsing.
12. Use predictive parsing to perform syntax analysis of control statements.
13. Discuss the theory of grammars.

14. Discuss syntax directed translation of control statements.
15. Write programs performing Lexical Analysis, Syntactic Analysis, and Code Generation of high level programming languages.

C. **Catalog Description:** This course studies the design and construction of compilers. Lexical analysis, syntactic analysis, and code generation are investigated in detail. Language design, interpreters, semantic analysis, intermediate code generation, and code optimization are also considered.

D. **Descriptive Overview of Course**

1. Outline of Course Content:

- a. Overview of compiling
 - (1) The phases of a compiler
 - (2) The lexical analyzer
 - (3) The syntactic analyzer
 - (4) The code generator
 - (5) Error analysis
 - (6) Example class language to be compiled
- b. Lexical analysis
 - (1) The role of the lexical analyzer
 - (a) Generation of a stream of tokens
 - (b) Creation of the symbol table
 - (c) Lexical error analysis
 - (2) Specification of tokens using regular expressions
 - (3) Recognition of tokens using transition diagrams or finite automata
- c. Syntax analysis
 - (1) Syntax definition through grammars and parsing trees
 - (2) Syntax analysis of assignment statements
 - (a) Types of errors
 - (b) Postfix notation
 - (c) The jump table algorithm
 - (3) Syntax analysis of control statements
 - (a) Top down parsing
 - (b) Recursive descent parsing
 - (c) Predictive parsing
 - (d) Other parsing techniques
 - (4) Introduction to the theory of grammars
- d. Code generation
 - (1) Syntax-directed translation
 - (2) Translation schemes and applications to assignment statements
 - (3) Translation of control statements
- e. Other possible topics
 - (1) Interpreters
 - (2) Semantic analysis
 - (3) Intermediate code generation
 - (4) Code optimization
 - (5) Compiler generators

2. Teaching Methodology: The lecture/discussion method will be used in this course. There will be a compiler project in which students will be expected to design and write a simple compiler. Some of this will be done in groups. Class lectures will stress methods for designing compilers. Handouts will occasionally be given to illustrate compiler design and components.

3. Text and Other Study Materials:

- a. Lecture text: Aho, A., Sethi, R., & Ullman, J. (1986). Compilers: Principles, techniques and tools. Reading, MA: Addison-Wesley.
 - b. Supplementary text:
 - (1) Aho, A., & Ullman, J. (1977). Principles of compiler design. Reading, MA: Addison-Wesley.
 - (2) Kruger, M. (Ed.) (1989). User's guide for VAX/VMS users at CUP, second edition.
 - c. Alternate text: Fischer, C., & LeBlanc, R. (1988). Crafting a compiler. Reading, MA: Benjamin/Cummings.
4. Methods of Evaluation and Assessment: The final grade will be determined as a percentage from the following evaluation methods with varying weights at the discretion of the instructor:
- a. Written examinations
 - b. Quizzes
 - c. Homework assignments
 - d. Programming assignments
 - e. Attendance
 - f. Performance

E. Supportive Library and Other Reference Materials:

- 1. 001.6425 A286p
Aho, A. & Ullman, J. (1977). Principles of compiler design. Reading, MA: Addison-Wesley.
- 2. 001.6425 B274c2
Barrett, W., & Couch, J. (1986). Compiler construction: Theory and practice, 2nd ed. Chicago: Science Research Associates.
- 3. 005.453 F529c
Fischer, C., & LeBlanc, R. (1988). Crafting a compiler. Reading, MA: Benjamin/Cummings.
- 4. 005.133 C12h49
Hendrix, J. (1988). A small C compiler: Language, usage, theory, and design. Redwood City, CA: M&T Publishers.
- 5. 001.6425 P998c, c.2
Pyster, A. (1980). Compiler design and construction. New York: Van Nostrand Reinhold.
- 6. 621.381958 S814i
Steele, D. (1978). An introduction to elementary computer and compiler design. New York: North-Holland.
- 7. 001.6425 Q145c
Waite, W. (1984). Compiler construction. New York: Springer-Verlag.