LSC Use Only
Number: _____
Action: _____
Date: _____

UWUCC Use Only
Number: #20
Action: _____
Date: _____

CURRICULUM PROPOSAL COVER SHEET
University–Wide Undergraduate Curriculum Committee

I. Title/Author of Change

Course/Program Title: CO 424
Suggested 20 Character Course Title: Compiler Construction
Department: Computer Science
Contact Person: Tia Watts or John Cross

II. If a course, is it being Proposed for:

XXX Course Revision/Approval Only
_____ Course Revision/Approval and Liberal Studies Approval
_____ Liberal Studies Approval Only (course previously has been approved by the University Senate)

III. Approvals

John A Cross — Department Curriculum Committee
Gary L Butenbaugh 8/15/90 — Department Chairperson
[signature] — College Curriculum Committee
[signature] — College Dean *
_____ — Director of Liberal Studies (where applicable)
_____ — Provost (where applicable)

*College Dean must consult with Provost before approving curriculum changes. Approval by College Dean indicates that the proposed change is consistent with long range planning documents, that all requests for resources made as part of the proposal can be met, and that the proposal has the support of the university administration.

IV. Timetable

Date Submitted to LSC: 8/15/90
to UWUCC: _____
Semester to be implemented: Spring 1991
Date to be published in Catalog: 1991-92

DESCRIPTION OF CURRICULAR CHANGE

I. Catalog Description - see attachment

II. Course Syllabus

A. Catalog Description - see attachment

B. Course Objectives

1. Students will demonstrate an understanding of the purpose and methods for formal specification of computer programming languages.

2. Students will know the functions of each phase of a compiler and will understand the relationships and interactions among these phases.

3. Students will be introduced to a variety of algorithms for implementing the front end phases of a compiler (Lexical Analyzer, Parser, Semantic Analyzer and Intermediate Code Generator).

4. Students will obtain hands-on experience in the design of a high-level, structured language, the specification of its grammar, and the implementation of its compiler.

C. Detailed Course Outline

1. Overview of the Compilation Process — 3 hours
   a. Phases and passes
   b. Lexical analysis
   c. Syntax analysis
   d. Semantic analysis
   e. Intermediate code generation
   f. Code generation
   g. Optimization
   h. Compiler development tools

2. Introduction to Grammars and Their Uses — 2 hours
   a. Context-free grammars
   b. Describing programming language features
   c. Handling attributes (meaning)

3. Lexical Analysis — 3 hours
   a. Recognizing lexemes
   b. Representing tokens
   c. Scanning techniques
   d. Regular expressions
   e. Available tools

4. Symbol and Literal Tables — 1 hour
   a. Organization and access
   b. Entry fields

5. Finite Automata — 2 hours
   a. Nondeterministic FA
   b. Conversion to deterministic FA
   c. Minimizing states

6. Error Handling — 1 hour
   a. Reporting the error
   b. Recovery techniques

7. Context-free Grammars — 3 hours
   a. Productions, alphabets and sentential forms
   b. Derivations and reductions

c. Ambiguity
d. Representing precedence
e. Controlling recognition
f. Influence of semantics
g. Influence of intermediate code form

8. Parsing — 3 hours
   a. Top-down and bottom-up
   b. Shift-reduce approach
   c. LR parser
   d. Parse table use
   e. Driver routine
   f. Parse table construction
   g. FIRST and FOLLOW functions
   h. Handling conflicts in the grammar
   i. Parse table generators

9. Syntax Directed Translation — 4 hours
   a. Grammar requirements
   b. Token-terminal association
   c. Synthesized and inherited attributes
   d. Parsing stack manipulation
   e. Type checking and coercion
   f. Semantic checking
   g. Overloading operators

10. Intermediate Code Generation — 4 hours
   a. Three-address code
   b. Quadruple operations
   c. Condition handling
   d. Using temporaries
   e. Handling declaration statements
   f. Backpatching forward references
   g. Array references

11. Storage Allocation — 1 hour
   a. Program code
   b. Identifiers (symbols)
   c. Literals
   d. Temporaries

12. Code Generation — 4 hours
   a. Object code form
   b. Register use
   c. Addressing modes
   d. Macro approach to generation
   e. Details of target machine
   f. Inefficiencies in the macro approach
   g. Backpatching forward references

13. Compilation of Procedures — 2 hours
   a. Symbol tables for block structures
   b. Call and argument quadruples
   c. Argument storage allocation and access
   d. Activation records and stacks

14. Optimization — 6 hours
   a. Placement of optimizing routines
   b. Peephole techniques
   c. Basic block analysis
   d. Flow graph representation
   e. Loop invariants

f. Induction variables
g. Next use and liveness of variables
h. Code transformations
i. DAG representation of basic blocks
j. Detecting common subexpressions
k. Programming situations that cause optimization problems
l. Parallel transformations

D. Evaluation Methods

Student grades are based on two in-class examinations, the final examination, and approximately four assignments.

Sample Assignments

In past versions of the course there have been four assignments which lead to the creation of a working compiler for a simple programming language. For each assignment, the students were divided into randomly selected three-person teams. Team membership was changed between assignments so that no student was put at a disadvantage for the whole term. Provisions were made so that teams that do not complete an assignment were not at a disadvantage for subsequent assignments. Each assignment is described briefly below.

1. Write a lexical analyzer for a given programming language. Each team writes a program that performs the lexical analysis functions for a simple programming language. The program output is used in project #3.

2. Write a grammar for a programming language.
Each team writes a grammar for a simple programming language. The grammar is fed into a parse table generator and evaluator program, which is provided to them. The resulting tables are used in project #3.

3. Write a syntax directed translator program.
Each team is provided with the driver program to which they must add the semantic analysis, error handling and intermediate code generation tasks. The output from the resulting program is used in project #4.

4. Write a code generator.
Each team writes a program that takes the intermediate code and produces an object file for given source programs. The object code from this file is then executed with a machine simulator that is provided to them.

E. Required Textbooks, Supplemental Books, and Readings

Suggested Text: Aho, Sethi and Ullman, Compilers: Principles, Techniques and Tools, Addison-Wesley, 1986. (new edition to be published in late 1990)

F. Special Resource Requirements

Students are not required to pay a lab fee or provide special materials or equipment for the course.

G. Bibliography

Fischer, Charles N., and LeBlanc, Richard J., Jr. (1988). Crafting a Compiler. Addison-Wesley.

Holub, Allen I. (1990). Compiler Design in C. Prentice-Hall.

Hunter, Robin. (1981). The Design and Construction of Compilers. John Wiley and Sons.

III. Course Analysis Questionnaire.

Section A: Details of the Course

1. This course provides computer science majors with an introduction to the structure of computer language translators in general and compilers in particular. It will be of particular interest to computer science majors planning to attend graduate school and those interested in systems programming. It is unlikely that this course would be of interest to students in other majors. The course is too specialized to be included on the General Education list.

2. No changes in existing courses are required.

3. The course follows the traditional approach of lecture courses in the Computer Science Department: lectures to cover the concepts and projects to provide practical experience. This course is novel only in that the projects are worked on in teams.

4. The course has been offered twice as CO 481, Special Topics, in Fall 1985 and Fall 1986. The syllabus in the attachment reflects the subject matter taught both times. Class sizes were 12 and 20. Student responses were encouraging and the results were very good. Students were able to understand the concepts and to apply them in the projects. We plan to offer it again in Spring 1991.

5. CO 424 is not a dual level course.

6. This course is not to be a dual-level course. At other universities this course may be a dual-level course, but IUP does not currently have a graduate program in Computer Science.

7. In many computer science degree programs, compiler theory or compiler construction is taught as an optional senior-level course for students interested in the systems programming area. Examples: CMPSC 420 at Pennsylvania State University and CS 122 at the University of Pittsburgh.

8. This course conforms closely to CS 15, Compiler Construction, in the ACM "Curriculum '78", CACM, Vol 22 No 3, March 1979, p. 153 and p. 156. ACM recommends CS 15; it does not require it.

Section B. Interdisciplinary Implications

1. CO 424 is designed to be taught by one instructor.

2. No corollary courses are required. CO 362, UNIX and C, may eventually be required or strongly recommended as a prerequisite.

3. There should be no duplication between the content of this course and that of courses from other departments. No course changes have been contemplated or discussed with other departments.

4. This course is not applicable to the Continuing Education program.

Section C. Implementation

1. All resources needed to teach CO 424 are currently available.

   a. Faculty

      One faculty member is required; one has been teaching the course. Two are current in the knowledge area of the course.

b. Space and Equipment

CO 424 requires the use of the Honeywell CP-6 system or the academic VAX mainframe and the VAX system in the Computer Science Laboratory or the AT&T system in the CS Laboratory. These systems are available.

c. No laboratory supplies are needed.

d. Library materials

Reference materials for the student's use during the programming projects are currently available.

e. No travel funds are needed especially for this course. However, this is a dynamic and rapidly evolving area of research and development. In general, ALL Computer Science faculty should attend at least one national conference a year. At least one of the faculty who are involved with this course should attend a conference which touches on this area at least once every two years.

2. None of the resources for this course are covered by a grant.

3. CO 424 will be offered once every two or three terms. Nothing about the course is designed for or restricted to offering the course during any particular semester; however, the course cannot be taught during the summer because of the complexity of the programming projects.

4. We anticipate having one section of CO 424 each time the course is offered.

5. Twenty students is probably the optimal number for one section; however, we will accommodate up to twenty-five. The instructor's ability to supervise and control students working on the complex projects, rather than facility availability, limits the number of students.

6. The pertinent professional societies do not spell out a recommendation, but the comment in CS is what other professors describe to us at national meetings.

7. This course will be an "upper-level elective". The addition of this course does not increase any degree requirements.

Section D: Miscellaneous

No additional data is attached.

LETTERS OF SUPPORT

No letters are attached because this course does not affect other departments.

CURRICULAR OFFERING/CHANGE AUTHORIZATION FORM - attached.

Catalog Description

CO 424 Compiler Construction 3C-0L-3SH

Prerequisites: CO 300 and CO 310

This course relates the formal concepts of automata and language theory to the practicalities of constructing a high-level language translator. The structures and techniques used in lexical analysis, parsing, syntax directed translation, intermediate and object code generation and optimization are emphasized.