TRAINING AIDS FOR TRANSLATOR DESIGN

by

JAMES R. MEYER

B.S., Benedictine College, KS, 1971

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

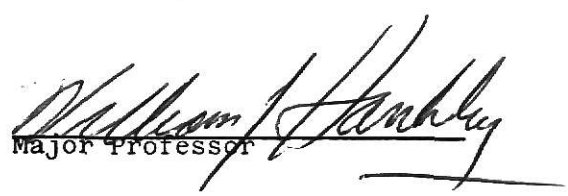
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1977

Approved by:



Major Professor

LD
2668
R4
1977
M49
c.2
Document

TABLE OF CONTENTS

FIGURES

FIGURES (continued)

CHAPTER 1

Introduction

The Interpreter Design Course, CS 286-700, is a first graduate level course in the concepts and design of compilers for computer systems. The course is designed to study the concepts, algorithms, and data structures of interpreters and compilers. The primary reference for the course is a text on compiler design. However, the text is used only as a supplement to the classroom presentations in the teaching of general concepts. The classroom presentations concentrate on teaching the course objectives using a student-developed interpreter as a model.

The model, the CS-700 Interpreter, was developed by students in the summer of 1975. The draft code covers the basic components of the interpreter, but requires further testing, debugging, and optimizing. In addition, the documentation is in varying degrees of perfection, and requires expansion and refinement. These discrepancies become individual and group projects for the students during the course.

The purpose of this project is to produce algorithms and training aids to be used in the classroom and as homework problems to support the teaching objectives. Specifically, they aid in teaching lexical scanning, top-down parsing, bottom-up parsing, and code generation. Another master's report, Models for Translator Design, Kansas State University, by Miles Tipton Clements Jr. concentrates on the execution of the CS-700 Interpreter. The abstract tutorial cases are designed to teach the concepts of a stack oriented sequential processor. The

narrative preceding the algorithms and training aids is designed to provide an introductory framework only, and no attempt is made to explain theories presented in the text. The specific implementation cases are designed to illustrate the concepts and prepare the students for the CS-700 Interpreter Projects. These specific implementation cases are also abstracts of the actual implementation. For example, the abstract may refer to an operator as "IPLUS", whereas in the real implementation an operator would have a numeric value. However, this level of abstraction lends clarity to details that are developing a concept.

CHAPTER 2

Lexical Scanning

2.1 General. Lexical scanning is logically the first function accomplished by a translator. In this phase the lines of source code are scanned and source code atoms are separated and identified. After an atom is separated and identified, a token is produced to represent the atom. In addition to producing tokens, some translators accomplish other actions during the lexical scanning phase. These actions may include semantic analysis and symbol table production. A symbol table performs a dictionary function and further describes the identifiers used in a source program. In the CS-700 Interpreter a symbol table is produced during lexical scanning to further describe identifiers in the source program.

2.2 Separation and Identification of Input. The scanning algorithm in Figure 1 demonstrates a typical scanner logic for separating and identifying source code atoms. This scanner is much like the scanner in the CS-700 Interpreter. In this algorithm an atom's type is determined by its first character. An atom may be typed as an identifier, an integer, a real number, an operator, a delimiter, a label, or a string. The right end of a variable length atom is determined when the scanner detects a blank, delimiter, or operator. If an atom is determined to be an identifier, then another algorithm matches it with a table of reserved words to make this differentiation.