

# Analysis, Design and Implementation of an Interactive Pascal Compiler on Personal Computers

This project is submitted in partial fulfillment of the  
requirements for award  
of the

DIPLOMA IN COMPUTER STUDIES  
of the  
HIGHER TECHNICAL INSTITUTE

CS/111

Project Supervisor : Mr. Christos Makarounas, Bsc  
External Assessor : Mr. Andreas Kattos

DESIGNED BY  
Flangofas John    Damianou Nikos

June 1994



---

## Summary

### *Analysis, Design And Implementation Of An Interactive Pascal Compiler On Personal Computers*

*Damianou Nikos*

*Flangofas John*

This project was initiated as an effort towards studying the sophisticated subject of compiler construction in depth. As of the very start, it was considered a very difficult achievement to have this project completed within time limits. The four months of work-time provided seemed rather inadequate to allow "wishful thinking". This of course was also due to the fact that apart from the compiler itself, an editing environment with a full implementation of menus, editing capabilities, mouse support, on-line help and more, was also due. This part, however, was finally dealt with after the completion of the compiler, using a powerful object-oriented language especially designed to handle interactive applications, the Turbo Vision facility of Turbo Pascal.

What puzzled us the most, was the building of the compiler. We had to start from scratch since we had no knowledge whatsoever on the subject of compiler-construction. The first thing to do was to find the right books and begin study on compilers in general, focusing on specific techniques and principles that govern the construction of a compiler. The aspects concerned were far too many since they involved knowledge in the following aspects :

- A third generation programming language with extremely good understanding of pointers, memory and recursion manipulation.
- A very good understanding of the programming language for which the compiler was to be built; that is Standard Pascal, which is one of the most strongly typed languages including a very complicated syntax to built a compiler for.
- An extremely good knowledge of machine or assembly language.

Based on hard study we managed to derive a model for our compiler and outline the phases and techniques that should be followed in implementing the compiler. This resulted in two important benefits that were the keys for the completion of the project. The first one was the division of the compiler into clear-cut phases and passes - we followed the multi-pass concept instead of the single-pass - and the second one was the decision to use the code-interpretation technique with the building of a pseudo-machine,

for the completion of the last part of the compiler : code generation. This meant that our compiler would not produce executable code, but we could then avoid using assembly or machine language for its building. Our knowledge and time available wouldn't have made it possible to complete the project if we wanted to output assembly or machine code. Finally, we solved the problem of the programming language by using Turbo Pascal for the implementation of the project.

The building of the Standard Pascal Compiler was definitely an extremely difficult task, but it was surely worth the effort.

---

# Contents

<b>Acknowledgments</b>	1
<b>Summary</b>	2
<b>Introduction</b>	4
<b>Part I : Understanding Compilers</b>	6
<b><i>Chapter 1 Programming</i></b>	6
1.1 History Of Programming .....	6
1.2 Programming Languages .....	8
1.3 The Need For High-Level Language Compilers .....	9
<b><i>Chapter 2 Compiler Construction</i></b>	12
2.1 Compiler Definition .....	12
2.2 Compiler Phases .....	15
2.2.1 Lexical Analysis .....	18
2.2.2 Syntax Analysis .....	20
2.2.3 Semantic Analysis .....	21
2.2.4 Intermediate Code Generation Phase .....	22
2.2.5 Code Optimization Phase .....	24
2.2.6 Code Generation Phase .....	26
2.3 Aims In Designing A Compiler .....	27
2.4 Compiler Construction Tools .....	30
2.5 Compiler-Like Tools .....	33

<b>Part II : Understanding Standard Pascal</b>	36
<i>Chapter 3 A Little Background</i>	36
3.1 The Pascal History .....	36
3.2 Description Of Pascal and Pascal-like Languages .....	37
<i>Chapter 4 Standard Pascal : The Language</i>	41
4.1 Pascal Features .....	41
4.1.1 Reserved Words .....	41
4.1.2 Standard Identifiers .....	41
4.1.3 Standard Functions .....	42
4.1.4 Standard Procedures .....	43
4.1.5 Operators And Symbols .....	43
4.1.6 Precedence Of Operators .....	45
4.2 A Pascal Program Example .....	45
<b>Part III : Designing the Compiler</b>	49
<i>Chapter 5 Basic Principles and Structure</i>	49
5.1 Logic Of the Language .....	50
5.2 Code Optimization .....	50
5.3 Memory Considerations .....	50
5.4 An Outline of the Compiler .....	50
<i>Chapter 6 Pass I : The Scanner</i>	56
6.1 Phase I : Lexical Analysis .....	56
6.1.1 Reading the Source Text .....	56
6.1.2 Format of the Intermediate Code .....	58
6.1.3 Building the Scanner .....	62
6.1.4 A Scanning Problem : Searching Words .....	71
6.1.5 Implementing the Symbol Table .....	80

6.1.6 Testing the Scanner .....	85
<b>Chapter 7 Pass II : The Parser</b> .....	92
7.1 Phase II : Syntax Analysis .....	92
7.1.1 Handling Symbol Input .....	92
7.1.2 The Grammar/Syntax Of Standard Pascal .....	95
7.1.3 Extended Backus-Naur Form : The Grammar Notation .....	95
7.1.4 Extended BNF Grammar/Rules Of Standard Pascal .....	100
7.1.5 Building the Parser : The Construction Rules .....	104
7.1.6 Design Principles and Techniques .....	119
7.1.7 Testing the Parser for correct Syntax Analysis .....	122
7.2 Phase III : Scope Analysis .....	122
7.2.1 Standard Pascal Scoping and Locality .....	123
7.2.2 Scope Analysis : The Logic .....	126
7.2.3 Scope Analysis : The Practice .....	133
7.2.4 Testing the Parser for correct Scope Analysis .....	136
7.3 Phase IV : Type Analysis .....	138
7.3.1 Distinguishing between Objects in a program .....	138
7.3.2 Standard Types : The simplest case .....	140
7.3.3 Dealing with Object Records for Constants .....	142
7.3.4 Dealing with Object Records for Variables.....	144
7.3.5 Dealing with Object Records for Arrays .....	151
7.3.6 Dealing with Objects for Records .....	153
7.3.7 Handling Type Analysis in Expressions .....	160
7.3.8 Dealing with Type Compatibilities in Statements .....	163
7.3.9 Performing Type Analysis for Procedures / Functions .....	166
7.3.10 Testing the Parser for correct Type Analysis .....	173
<b>Chapter 8 Code Generation : The Last Phase</b> .....	175
8.1 Defining the "Object" Code : How it works .....	176

8.1.1 The Code Format .....	181
8.1.2 Managing Variables .....	182
8.1.3 Managing Expressions .....	187
8.1.4 Managing Statements .....	193
8.1.5 Managing Standard Functions and Procedures .....	200
8.1.6 Managing Pointers and Files .....	201
8.1.7 Managing Procedure and Function Calls .....	203
8.2 Generating the "Object Code : The practice .....	209
8.2.1 Generating Code for Variables / Parameters .....	210
8.2.2 Generating Code for Expressions .....	213
8.2.3 Generating Code for Statements .....	216
8.3 Pass III : The Assembler .....	216
8.3.1 Tackling Forward References .....	216
<b>Chapter 9 The Editor</b> .....	<b>222</b>
<b>Conclusion</b> .....	<b>224</b>
<b>Appendixes</b> .....	<b>226</b>
Appendix A Pseudocode for the Administrator .....	226
Appendix B Pseudocode for the Scanner .....	228
Appendix C Pseudocode for the Syntax Analysis .....	239
Appendix D Pseudocode for the Scope Analysis .....	255
Appendix E Pseudocode for the Code Generation .....	259
Appendix F Pseudocode for the Assembler .....	264
Appendix G Pseudocode for the Code Interpreter .....	268
Appendix H Error Messages .....	270
Compilation-Errors .....	270

Runtime-Errors .....	278
Appendix I Standard Pascal Syntax Diagrams .....	280
<b>References</b>	<b>290</b>
<b>Glossary</b>	<b>292</b>