PREFACE

There are a number of fine textbooks on the subject of data structures. While writing this text, we found ourselves referring to many of them, as to their contents, manner of presentation, and algorithms applied to develop the data structures. Over the years, we have used some of these texts as required reading for our classes. We found them very informative and interesting.

The problem is that our students did not. On year-end student evaluations, we would find that the question "How would you rate the textbook used?" generally yielded responses of *Poor* or *Unsatisfactory*, many times accompanied by comments which included terminology unsuitable for print. All too often, students would complain about the text to us, department chairpersons, the unfortunate administrator being punished with the responsibility of listening to student complaints, and to other students. Our perfectly reasonable response was "But it is the best selling text in the area, written by a genius in a style which is generally considered as Shakespearean in nature". This apparently only served to further anger students.

It was the oft recited comment "I never bothered reading the textbook after the first chapter and relied entirely on your notes" that made us think, and made us eventually write this text.

The students that we teach are *not* computer science majors. They do *not* (generally) come to us having completed two or more courses in structured programming, multiple courses in higher level mathematics, computer organization, predicate calculus, and Gate's-only-knows-what other technical courses computer science courses are required as prerequisites. While some of our students do indeed feel compelled to spend all of their waking hours at a computer, most merely depend on it as a tool, using it as needed.

The students we generally accept into our classrooms are Computer Information Systems majors, other business majors, engineering majors, liberal arts students who are required to take a course in 'Computers', undergraduate and graduate students who need an understanding of computer functioning (but do not wish to devote their lives to developing algorithms which will calculate pi to near infinity), and any number of other 'unenlightened souls' (from a computer science perspective). They have an understanding of bits and bytes, but not how they actually relate to computer functioning. Their exposure to these topics is founded in introductory courses wherein instructors force them to learn the terminology, and perhaps to convert between decimal and binary, purely for the sadistic pleasure gained through student suffering. These students have generally had one course in programming, usually COBOL or BASIC or Pascal; enough to allow them to write simple programs, but not enough to manipulate code to make it dance to the music. Some of students are actually virtuosos at programming, but without any real understanding of how or why the code does what it does.

It is an audience which is growing proportionately with the number of individuals who are using, and even relying upon, computers in their everyday lives. It is an audience which appreciates the tenet that in order to advance their abilities, they must first understand how a computer functions.

This textbook is intended for this audience. It is intended for those individuals who had have some exposure to computers and are able to use them as tools and, most likely, enjoy and appreciate their potential. It is intended for those individuals who *want* to understand how a computer functions so that they can more fully exploit it for their purpose. In short, it is intended for individuals who have a desire to use *and understand* computers, as they exist now, as well as possible. It is NOT intended for people who want to build computers, or extend computer functioning beyond what it is presently capable of. Those individuals are best advised to refer to any of the great textbooks we referred to in the beginning of this preface.

Initially, we intended titling this text *Data Structures for Dummies*, given the popularity of the approach. We couldn't do it. We have tried to keep our explanations and discussions as simple as possible, but try as we might, we couldn't make it understandable to a *dummy*. Then again, we really didn't think we should, nor do we feel that our students are dummies. This is not a topic for *dummies*. It is a topic for relatively smart people who want to be better informed on computer functioning and how to better use the computer. It is not possible to be a *dummy* and have these aspirations.

The text takes a very *primitive* approach, especially in the early chapters. *Primitive*, in the manner we are using it, implies that we take a nuts-and-bolts approach to data structures, as opposed to beginning at a higher level of abstraction. We start off by showing how individual bits, and the grouping of bits as bytes, are used to store and interpret data. It is a theme which we underscore throughout the text, although the emphasis lessens as the text progresses. We think it is an important theme, since we have found that students can relate to higher levels of abstraction if, and frequently only if, they can understand the underlying mechanisms. We try not to take for granted the student's interest in the mechanics behind the concepts, so that they can more fully appreciate the abstract data types which we discuss.

It is also for that reason that we have chosen the c programming language to illustrate the data structures we develop. C is a very primitive language. It is very easy to make mistakes. Big ones. We do not see this as necessarily bad. While making mistakes may not be good, it is a great way of learning how to avoid them in the future. We not only encourage students to make mistakes, but even show them how to make them, and why they made them, throughout the text.

C is also an extremely powerful language. One of the central axioms we apply throughout the text is:

> *"Give me an address, tell me the type of data which is stored there, and I will give you the value of that data."*

In a nutshell, that is all there is to data structures. Simple enough, but there are some other considerations:

1. How do I determine where the data is stored?
2. What type of data should be stored there?
3. How do I analyze the data which is stored there? And,
4. How can I most efficiently store, retrieve, and utilize the data I will find at a given address?

These are the questions this text attempts to deal with. Starting with the most basic of data types, we try and address the considerations listed above. As we begin developing new data types, we try not to lose sight these issues.

The text is divided into four sections, five if we count our (brief) introduction to data structures using object oriented programming:

1. **Section 1** (Fundamental Concepts) is intended as a remedial course for non-computer science majors. That is not to imply that the subject matter is necessarily simplistic, but merely that most computer science majors would be well acquainted with the material before entering this course. As we state in the introduction to the section, it is intended to *level the playing field*.

2. **Section 2** (Common Abstract Data Types) is generally where data structures texts for Computer Science majors start. The heading is not deceptive, however. The data structures covered in this section (Numeric Arrays, Strings, and Structured Data Objects) are the most commonly used structures. Additionally, we overview some of the ways in which these data structures can be readily manipulated (Stack, Queues, Heaps, and file processing).

3. **Section 3 (**Issues of Concern and Data Types to Deal With Them) introduces some of the limiting issues in programming and data structures (linked lists) which are intended to deal with them. We address the major limitations (a fixed number of contiguous bytes of RAM) and show how we can overcome them. We also introduce one major trade-off which programmers encounter: The ease of creating and maintaining non-ordered lists vs. the speed of finding and retrieving data from ordered lists.

4. **Section 4** (Hierarchical Data Structures) offers new data structures (Binary Trees, AVL and Btrees) and techniques (recursion) which are intended to deal with the dilemmas discussed in the previous section. Frequently, in a number of texts, these topics come under the heading Advanced Data Structures. We avoid this designation because we feel that the student should not be intimidated into thinking these approaches are complex.

5. **Section 5** (An Introduction to Object Oriented Data Structures) is intended as just that: An Introduction. For the reasons given above, we have decided to adopt a primitive approach toward our discussion of data structures, and have used the C Programming to that end. However, given the increased emphasis on object oriented programming, and the interesting data structures which the approach uses, we felt that, at very least, these structures could not go unmentioned.

We trust we have approached our goals, and welcome comments on our successes and failures.