Parsing Foundations

A Study of Data and Parsing

Unfinished Draft of First Edition

Parsing Foundations

A Study of Data and Parsing

Unfinished Draft of First Edition

Richard R. Masters
Seattle, WA

Self Publishers Worldwide
Seattle San Francisco New York
London Paris Rome Beijing Barcelona

This book was typeset using LaTeX software.

Preface

This book was created from a desire to make parsing easier to understand. Parsing is an important topic in the field of computer science and the author struggled to learn about parsing. It was not that the ideas themselves were particularly difficult, but that it required a huge investment of time and some money to identify, locate, acquire, organize, and understand the relevant information. In many cases, the information presented assumes the reader is fluent with set-builder notation[1] and sometimes particular symbols may not be defined because they have been used in previous papers on the subject that the reader is assumed to have read.

It became clear that it would be a lot easier to learn the subject from a book that covers the relevant topics with through explanations. It was also clear that there is a lot of existing research, some of it several decades old, that could benefit from a wider audience. Finally, a good understanding of parsing rests on a good understanding of data, which is what a parser acts upon.

Prior Work

Prior to the writing of this book, there were not that many resources for learning about parsing that were both freely available and of high-quality. There were no more than a handful of books about parsing on the market and the most relevant to a general scientific audience was a relatively expensive book about compilers[2]. There are also quite a few papers and articles written for academic journals that are written for a narrow audience by use of vocabulary and notation. Finally, while there are some good online resources, they are lost within a sea of information that is either outdated or poorly produced, or only covers a limited set of topics.

Design Goals

This book is designed to be as simple to understand as possible. This book is built from the ground up with relevant topics, starting from the basics and building up to advanced topics, using a simple and intuitive style of presentation.

Some of the explanations in this book may seem excessively long so it is worth noting that keeping explanations short is a much lower priority for this book than providing a thorough explanation. The idea is that far more time is wasted by avoiding difficult material, skimming it repeatedly, and finally focusing your mind enough to understand it, than is wasted reading and understanding material that is excessively explained.

Audience

This book can serve as an introduction to topics on data and parsing for the working software developer or student of computer science. The book covers foundational topics of parsing using concrete examples and common tools used in the field. In an academic setting, the book would serve well as a companion book for courses that only cover parsing as part of a larger topic, such as compiling.

Uses for this Book

The book may also be of interest to you if you are an educator, a blogger, or if you are writing a book. This document is licensed under the Creative Commons Zero v1.0 Universal license[3]. You are free to copy this content, distribute

[1] `https://en.wikipedia.org/wiki/Set-builder_notation`
[2] `https://en.wikipedia.org/wiki/Compilers:_Principles,_Techniques,_and_Tools`
[3] `https://creativecommons.org/publicdomain/zero/1.0/legalcode`

it, include it in your own content, or sell it, and you may do so *without attribution*. Just copy what you want and move on.

Terms and Symbols

Many of the ideas covered in this book originated in the 1950s and 1960s, near the dawn of the computer age. In the decades since, researchers and educators have defined quite a few names for various concepts surrounding data and parsing. They have also adopted numerous symbols and notations as a convenient shorthand to write down their ideas. While it can be tedious to learn (and explain) these names and symbols, it is necessary if you want to be able to follow discussions or read original research papers on these topics. In some cases, words that you may be familiar with in common language may have a different definition in academic papers. This can be confusing if you are not already familiar with that term. Therefore, in this book, we will error on the side of explaining the academic terms and symbols when they are used. Sometimes it takes a number of definitions conspiring together to explain a larger concept, so bear with us if it seems like we are just defining words for no apparent reason.

Prerequisites

For the chapters on data, the reader should have a basic knowledge of computers, such as knowing about a monitor, mouse, disk, file, and an application.

Table of Contents

Chapter 1

Data

1.1 Symbols

The signs, sounds, marks, and words humans use to represent ideas are called *symbols*. For example, the twenty six letters of the English alphabet are symbols representing sounds or parts of words. Words are also symbols for the ideas they represent. The numerals 0 through 9 are symbols for numbers. As long as something represents, or *symbolizes* something else, it is a symbol. We use symbols as sort of "real world" currency to remember or communicate something meaningful to ourselves or others.

1.1.1 Interpreting Symbols

Wikipedia defines *data* as any sequence of symbols for which "a specific act of interpretation" gives those symbols meaning[1]. In other words, the essential nature of data is that is in a *sequence*, which means that the symbols occur one after another, and that there is a particular meaning associated with the ordering and choice of the symbols in the data.

1.1.2 Manipulating Symbols

Computers are well known for their ability to manipulate data. A computer can read and *execute*, or follow, instructions as represented by a sequence of symbols. These instructions can, in turn, direct the manipulation of symbols stored in the computer's memory. It can be confusing to think of a computer working with the kind of visual symbols that humans typically think of like a + (plus) sign or the numeral 2, and that would be because they do not actually work directly with those kinds of symbols. The symbols that a computer can read, follow, and manipulate are quite limited. In fact at the lowest levels, there are only two symbols. Inside the computer, these symbols exist as electrical states inside the transistors and wires of the computer. From an electrical perspective, we think of these states as *off* and *on*, but from a numerical perspective these states symbolize 0 and 1, respectively.

1.2 Binary Data

1.2.1 Bits

A numerical symbol which is either 0 or 1 is known as a *binary digit*, or *bit* for short. At the most fundamental level, bits are the letters of a very simple alphabet that both humans and computers can manipulate in an equivalent way. For example, both humans and computers are capable of numerically adding together the symbols 1 and 0 to produce the symbol 1. It is through these common definitions and ways of processing symbols that computers can perform computations that are relevant to humans.

[1] `https://en.wikipedia.org/wiki/Data_(computing)`

1.2.2 Binary Codes

Since computers are only able to manipulate sequences of bits, we have devised an incredibly rich variety of ways, known as *codes*[2] to map the information available to us into sequences of ones and zeros. For example, *decimal*[3] numbers, which are the standard numbers we use from day to day, can be converted to a binary number composed of binary digits, using a *binary code* as seen in Table 1.1.

Number	Bits
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 1.1: Mapping of Numbers to Bits

1.2.3 Nibbles

A *nibble* is four bits.

1.2.4 Hexadecimal

Hexadecimal is a numerical encoding that allows the use of 16 different symbols for each digit, 0 through 9 and A through F. Each symbol encodes 4 bits of a binary number.

1.2.5 Bytes and Octets

A *byte* is typically 8 bits. An *octet* is always 8 bits.

1.2.6 Exercises for Section 1.2

Exercise 1.2.6.1: Write the numbers 8 through 15 using 4 bits.

Exercise 1.2.6.2: Create a mapping from the numbers 0 through 8 to two digits of the three symbols 0, 1, and 2.

1.3 Character Encodings

Characters are symbols that represent sounds or parts of words.

1.3.1 ASCII

1.3.2 Terminals and Control Characters

1.3.3 Locales

1.3.4 Unicode

The Unicode® Standard is a universal character encoding standard for all written characters and text.

1.4 Summary of Chapter 1

[2] https://en.wikipedia.org/wiki/Code
[3] https://en.wikipedia.org/wiki/Decimal

Chapter 2

Parsing

Parsing as a way of interpreting data based on how it matches *structural patterns*, such as a *formal grammar*.

Index