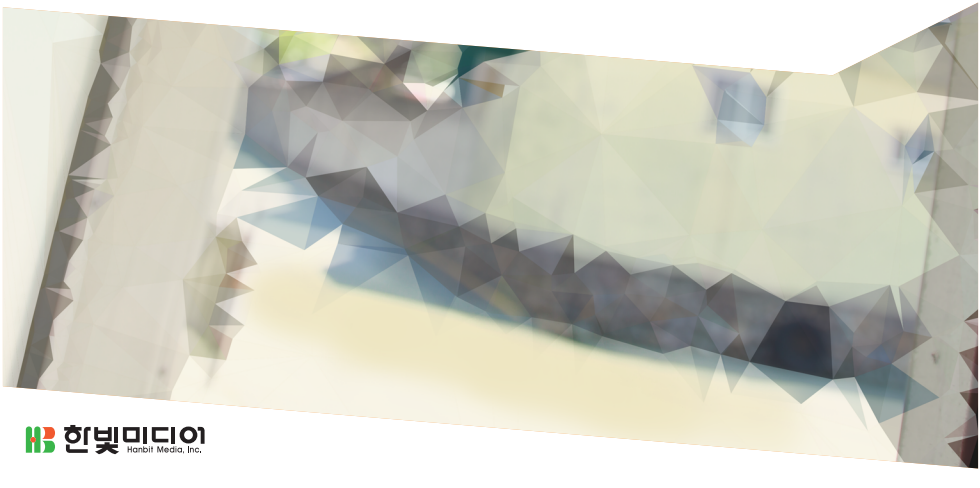
Hanbit RealTime 112

OCR 프로그래밍

이미지 인식

김세훈 지음



한빛미디어 Hanbit Media, Inc.

OCR 프로그래밍 이미지 인식

김세훈 지음

한빛미디어 Hanbit Media, Inc.

표지 사진 **김세훈**

이 책의 표지는 저자 김세훈 님이 보내 주신 풍경 사진을 담았습니다.
리얼타임은 독자의 시선을 담은 풍경 사진을 책 표지로 보여주고자 합니다.

사진 보내기 **ebookwriter@hanbit.co.kr**

OCR 프로그래밍 이미지 인식

초판발행 2015년 8월 21일

지은이 김세훈 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-773-6 15000 / **정가** 13,000원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 정지연 / **교정** 이미연
디자인 표지/내지 여동일, 조판 최송실
마케팅 박상용 / **영업** 김형진, 김진불, 조유미

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea
Copyright © 2015 김세훈 & HANBIT Media, Inc.
이 책의 저작권은 김세훈과 한빛미디어(주)에 있습니다.
저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ **김세훈**

"내가 지금 뭐 하고 있지?" 2010년 40살이 되어 불현듯 나를 돌아봤다. 열심히 살려고 부단히 노력했던 것 같은데, 회사와 집만을 오가며 답답한 삶을 살고 있음을 느꼈다. 그래서 다짐했다. "이제부터 즐기면서 살자." 그런데 삶은 녹록하지가 않았다. "되는 일보다 안되는 일이 더 많다. 그것이 인생이다."라고 하시는 아버지의 말씀도 진리처럼 느껴졌다. 몇 년이 흐른 후 내가 변해 있음을 느꼈다. 무엇이 나를 변화하게 했을까? 그건 지금까지 5년 넘게 매일같이 인생에 대한 글을 쓴 일이었다. 청소년기에 가치관이 정립된다고 학교에서는 가르치지만, 나는 죽을 때까지 가치관을 정립하며 죽을 때도 깨우치지 못한 진리들이 많을 것이다. 좋은 말씀을 써 오라는 숙제를 받은 초등학생 조카에게 필자는 말했다. "일기 쓰면 인생이 변한다." 하루의 일과가 아닌, 인생의 진지한 글들을 써 보세요. 시간이 한참 흐른 후 자신이 변해 있음을 발견하게 됩니다.

필자는 현재 '지식소프트' 1인 회사에서 부장으로 재직 중이다.

언어를 배울 때 사전을 찾아본 적이 있을 것이다. 사전에서 원하는 단어를 찾는 방법은 간단하다. 정렬된 순서에 의해서 찾고자 하는 단어가 일치하면 찾은 것이다. 인식 프로그램의 가장 대표적인 것이 전자사전인데, 전자사전에서 단어를 찾는 것도 일반 사전에서 단어를 찾는 것과 마찬가지다. 데이터베이스가 구축되어 있고, 찾고자 하는 단어를 찾으면 된다. 모든 인식 프로그램이 이 원리를 따르고 있다. 그렇다면 이미지나 소리의 인식은 무엇이 다를까? 프로그램에서 새로운 데이터 타입을 만들고 사전처럼 기존에 만들어진 데이터와 비교할 때 정확한 일치가 어려우므로 확률을 사용한다는 점이다. 즉, 일치하는 확률이 높은 것을 결과값으로 결정하며, 얼마나 정확한 결과값을 도출하느냐가 인식률의 높고 낮음을 나타낸다.

문자 인식에 대한 기술은 오래전에 나왔으며, 이제는 소리 인식으로 넘어갔다. 그러나 기술 공개는 이루어지지 않고 있다. 이 책에서 설명하는 내용은 필자가 생각해낸 방법이므로 이 방법만이 정답은 아니다. 하지만 개인적으로 이 방법이 문자 인식OCR을 하는 최적의 방법이라 생각한다.

데이터 타입이 무엇인지는 'How-to Series' 1편인 『소수와 RSA 알고리즘으로 배우는 Big Number 연산 - 구조체와 자료구조의 이해』(한빛미디어, 2013)에서 설명하였으며, 프로그래밍의 시작은 이 데이터 타입을 이해하는 것이라고 본다. 이 책에서도 문자 이미지를 담기 위한 데이터 타입을 가장 먼저 다룰 것이다. 여기서 다루는 새로운 데이터 타입을 이해하면 왜 한글 인식이 영문 인식보다 어려운지를 알게 될 것이다.

그다음으로는 문자 이미지를 어떻게 데이터 타입에 넣는지를 다룬다. 이는 문자

인식에서 가장 어려운 부분이다. 하나의 문자 이미지가 문자 데이터 타입에 담기면 기존에 만들어 놓은 기본 데이터 타입과 비교하면 된다. 전체 이미지에서 하나의 문자 이미지만을 가져오는 방법은 이미지마다 다르게 적용될 수 있기에 이미지를 분석하는 기술을 요구한다.

이미지 인식의 마지막은 이미지 변형Image Transform이다. 이미지 변형은 포토샵 같은 프로그램에서 주로 사용하는데, 간단한 이미지 변형으로 인식률을 어떻게 높이는지 알게 될 것이다. 이미지 변형에서는 수학적인 내용이 포함되나 중학교 수준의 삼각함수를 이해하고 있는 독자라면 쉽게 이해할 수 있다.

'How-to Series'는 프로그램 언어를 설명하는 책이 아니며, 프로그램을 어떻게 만드는지를 독자와 함께 공부하려는 목적으로 만들었다. 이 책을 가장 효과적으로 보려면 전체를 한 번 빨리 훑고 직접 프로그램을 만들면서 막히는 부분은 이 책을 보면서 풀어나가는 것이 좋다.

'How-to Series'는 1년에 한 가지 주제로 펴내고 있다. 책마다 개별 주제로 만들어지지만 먼저 출간된 책에서 설명한 내용을 이후의 책에서는 생략하는 경우가 있다. 시리즈 3편인 이 책에서 구현하는 프로그램은 MFC를 사용하는데, MFC는 『MFC 프로그래밍 : 주식 분석 프로그램 만들기』(한빛미디어, 2014)에서 상세히 설명하였기 때문에 이 책에서는 자세한 설명을 생략하였다. 예를 들어, '도구 상자'를 사용하여 버튼을 만든다거나 클래스를 추가하는 방법 등이 생략되었다. MFC에 대해 충분히 이해하고 있지 않다면 『MFC 프로그래밍 : 주식 분석 프로그램 만들기』를 참조하면서 보기를 권한다.

세 번째 책인 『OCR 프로그래밍: 이미지 인식』은 힘들 때 항상 함께해 준 나의 친구 '박흥준'에게 바친다. 또한, 나의 사랑하는 조카 '신효인'에게 선물로 준다. 나의 영원한 모델인 효인이의 개인 사진사로 활동하면서 조금 더 사진에 대해 관심을 갖게 되었고, 이미지 인식에 쉽게 접근할 수 있었다고 생각한다.

대상 독자 및 참고사항

이 책은 C++를 알고 있는 초·중급자를 대상으로 하며 Windows 환경에서 MFC로 OCR 프로그램을 만드는 방법을 설명합니다. 주제를 가지고 프로그램을 배우자는 목적으로 OCR이라는 주제로 처음부터 단계적으로 프로그램을 만들어 갑니다. 사진에서 문자를 인식하는 OCR 기술은 다양한 프로그램으로 응용될 수 있는 기술이며 교육의 목적으로 Windows 환경에서 구현되었지만, 동일한 원리로 스마트폰 애플리케이션에서 응용할 수 있습니다.

이 책에 수록된 코드들은 Windows에서 생성하여 실행해야 하며, Visual Studio 2008에서 구현되었습니다.

예제 테스트 환경

사용 프로그램	설명
Visual Studio 2008 이상	예제는 Windows 환경에서 테스트하였다.
Windows OS	Windows 환경에서만 실행할 수 있다.

예제 파일은 다음에서 다운로드할 수 있습니다.

- https://www.hanbit.co.kr/exam/2773

'How-to Series'는 방법론적인 부분들에 초점을 맞추어 만들어진 책입니다. 일방적인 지식의 전달이라기보다는 "이러한 방법도 있구나"하고 이해하는 것이 중요합니다.

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾기도 쉽지 않습니다. 또한, 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1 eBook First – 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한, 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2 무료로 업데이트되는 전자책 전용 서비스입니다

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3 독자의 편의를 위해 DRM-Free로 제공합니다

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 어려운 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

지금 보시는 전자책에 소유 권한을 표시한 문구가 없거나 타인의 소유권한을 표시한 문구가 있다면 위법하게 사용하고 있을 가능성이 큽니다. 이 경우 저작권법에 따라 불이익을 받으실 수 있습니다.

다양한 기기에 사용할 수 있습니다. 또한, 한빛미디어 사이트에서 구매하신 후에는 횟수에 관계없이 내려받으실 수 있습니다.

한빛미디어 전자책은 인쇄, 검색, 복사하여 붙이기가 가능합니다.

전자책은 오탈자 교정이나 내용의 수정·보완이 이뤄지면 업데이트 관련 공지를 이메일로 알려 드리며, 구매하신 전자책의 수정본은 무료로 내려받으실 수 있습니다.

이런 특별한 권한은 한빛미디어 사이트에서 구매하신 독자에게만 제공되며, 다른 사람에게 양도나 이전은 허락되지 않습니다.

차례

chapter 1 OCR이란 무엇인가? 013

1.1 사진에서 문자 이미지 가져오기 014
1.2 연산이 가능한 데이터 만들기 016
1.3 확률을 이용한 인식 019
1.4 영문이 한글보다 인식률이 높은 이유 021

chapter 2 기본 이미지 데이터 만들기 023

2.1 새로운 데이터 구조체 027
2.2 기본 이미지 데이터 029
2.3 문자 이미지 추출하기 035
2.4 이미지 데이터 만들기 067
2.5 기본 이미지 데이터 불러오기 074

chapter 3 문서 이미지 OCR 089

3.1 확률을 이용한 OCR 093
3.2 줄바꿈 문자 추가하기 101
3.3 공백 문자 삽입하기 105
3.4 특수 문자 추가하기 116
3.5 마침표, 쉼표, 따옴표 인식 121
3.6 대문자와 소문자 보정 130
3.7 문자 색의 범위 조정으로 인식률 높이기 139

chapter 4 자동차 번호판 인식 143

4.1 사진 크기 변경 및 흑백 사진 만들기 144
4.2 문자 이미지 추출 151
4.3 문자 이미지 데이터 변환 163
4.4 번호판 인식 168
4.5 이미지 데이터 변형 175

부록 187

A.1 사진 크기 변경하기 187
A.2 사진 회전하기 190
A.3 흑백 사진 만들기 192
A.4 서명 넣기 195

글을 마치며 201

chapter 1

OCR이란 무엇인가?

인식은 인공지능AI의 한 분야라고 볼 수 있다. 그 시작은 사전과 같은 간단한 것이나 궁극적으로 추구하는 것은 사람 간의 의사소통일 것이다. 지리적인 문제와 다양한 언어는 인간의 의사소통을 단절시켰지만, 인공지능은 기계를 통한 연결로 다른 지역이나 다른 나라 사람과의 의사소통을 원활히 하기 위해 발전해 왔다.

그런 면에서 음성 인식은 많이 발전한 듯 보이지만 아직 시작 단계다. 그렇다면 이미지 인식은 어떨까? 이 책에서 다루려는 이미지 인식은 문자에 국한되고 일반적인 사물을 인식하는 것은 현재로써는 어렵다. 하지만 2차원적 이미지 인식도 음성 인식과 별반 다르지 않다. 그런데도 음성 인식보다 이미지 인식이 먼저 발전한 이유는 무엇 때문일까? 일정한 틀 안에 이미지를 담기가 더 쉬웠기 때문이다. 문자를 나누기가 조금 더 쉬웠고 프로그램 언어로 표현하는 방법이 용이했다.

1999년에 처음으로 문자 인식 펜을 본 적이 있다. 종이에 인쇄된 알파벳을 따라 그으면 단어의 뜻이 펜 표면에 뜨고, 스캔한 글이 인식되어 워드 파일로 만들어졌다. 이후 문자 인식OCR, Optical Character Recognition은 더 발전하고 많이 보편화되었다. 예를 들어, 주차장에서 카메라로 자동차 번호판을 읽는 것도 문자 인식이다.

이 책의 마지막에 자동차 번호판 인식을 다루지만, 일반 카메라로 찍은 사진이기에 실제 환경과는 다른 방법으로 인식할 것이다. 시중의 자동차 번호판 인식은 일반 카메라가 아닌 특수 카메라로 찍기 때문에 문자 부분과 바탕 부분이 명확히 흑백으로 구분되어 인식하기가 훨씬 용이할 수 있다.

이 책을 통해 OCR을 이해하고 응용한다면 일정한 형태의 문자를 인식하는 방법을 쉽게 찾을 수 있을 것이다.

1.1 사진에서 문자 이미지 가져오기

사진에서 문자를 인식하기 위해 가장 먼저 해야 할 일은 문자 이미지 부분을 알아내는 것이다. 이 책에서는 이를 파싱한다고 부를 것이다. 이제부터 이미지 파싱이라고 하면 이미지에서 문자 부분을 떼어 낸다고 이해하면 된다. 문자 이미지는 사각형으로 따낼 것이며 2차원(평면)의 사진에서는 색으로만 구분하기 때문에 정해진 문자 색을 기준으로 문자 이미지를 인식할 것이다. 또한, 문자 색으로 지정된 특정 색 이외의 것은 모두 바탕으로 생각하고 무시하면 된다. 즉, 문자 색이 검은색이라면 사진에서 검은색 이외의 부분은 모두 흰색이고, 사진 한 장에는 검은색과 흰색만이 존재한다고 보면 된다. 즉, 사진을 두 가지 색으로만 나누어 문자와 바탕으로만 생각한다. 흰색 종이 위에 다양한 색의 문자들이 있다면, 이때는 흰색을 제외한 나머지 색을 검은색으로 바꾸어 두 가지 색으로만 이루어진 이미지를 만들면 된다.

파싱의 기본은 문자와 바탕이라는 두 가지 색으로만 이루어진 이미지를 만드는 것이다. 하지만 실제 프로그래밍에서는 색을 바꾸는 작업은 하지 않을 것이다. 이 작업도 CPU의 많은 작업을 필요로 하기 때문에 프로그램의 속도를 늦출 뿐이다. 다음 그림은 이미지에서 문자를 인식하기 위해 어떻게 변환 작업을 하는지를 보여준다.

[그림 1-1]은 문자 색을 기준으로 나머지를 바탕으로 만들었고, [그림 1-2]는 바탕 이외의 색을 모두 문자로 판단하여 변환하였다. 이때 중요한 것은 변환한 후의 이미지는 검은색 문자와 흰색 바탕으로만 이루어졌다는 것이다. 무엇을 문자로 인식하느냐에 따라 파싱의 처음 작업은 다양하게 이루어질 수 있다.

그림 1-1 문자 색 기준의 이미지 변환

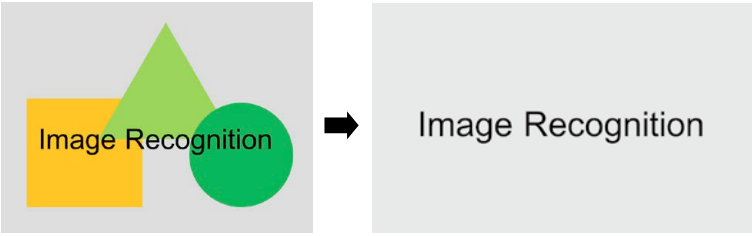
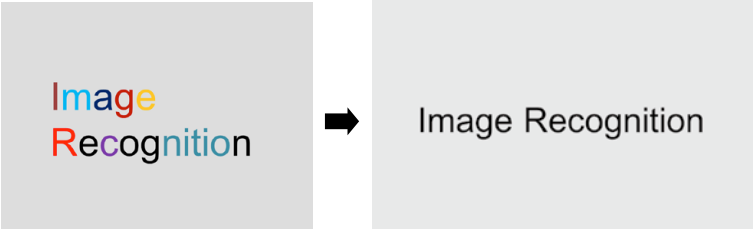


그림 1-2 바탕색 기준의 이미지 변환



앞에서와 같이 인식할 문자가 명확하다면 파싱 작업을 하여 개별 알파벳으로 나누면 된다. 'Image Recognition' 단어는 16개의 알파벳으로 나누면 되는데 나눈 문자는 [그림 1-3]과 같이 개별 이미지에 꽉 차게 보일 것이다. 즉, 검은색 부분을 기준으로 나누어 사각형의 테두리에는 항상 검은색이 존재하게 된다.

문자를 하나씩 나누는 작업은 이미지마다의 특성을 분석해야 한다. 실제 이미지는 기울어져 있을 수도 있고, 사진 속의 이미지라면 사진을 찍는 각도에 따라 이미지가 틀어져 있을 수도 있다. 하지만 이 책에서는 쉽게 이해하기 위해 깨끗한 이미지를 예로 사용하여 쉬운 파싱을 할 것이다. 문자 인식에서 가장 어려운 부분이 파싱인데, 다행히도 문자와 문자 사이는 겹친 부분이 없기에 파싱이 용이하다. 각각의 문자가 겹쳐 있다면 인식이 불가능할 수도 있다.

[그림 1-3]과 같이 하나씩 나누어진 문자 이미지는 인식을 위해 크기가 같은 이미지로 변환해야 한다. 모든 문자는 비교할 수 있는 일정한 틀 안에 담아야 하기 때문이다.

그림 1-3 하나의 문자로 나누기

[그림 1-4]는 가로, 세로의 크기가 같은 사각형 안에 각 알파벳을 담은 것이다. 하나의 문자는 늘려질 수도 있고 축소될 수도 있다. I[아이]는 사각형 안에 꽉 차게 된다. 그렇다면 소문자 l[엘]은 어떻게 될까? 이 책에서는 문자 형식을 Arial Type으로 진행을 하기 때문에 대문자 I[아이]와 소문자 l[엘]은 같은 이미지로 나타난다. 이러한 경우에는 문자가 처음 시작하는 문자인지를 판단해야 하며 프로그램으로 이 부분을 해결해야 한다.

그림 1-4 크기가 같은 문자 만들기

문자 인식 프로그램에서는 [그림 1-4]와 같이 이미지 자체를 일정한 크기로 변환하는 작업을 하지 않는다. 이러한 작업은 속도에 영향을 주는 불필요한 작업이므로 문자 인식 프로그램에서는 이미지의 변환 없이 바로 연산이 가능한 데이터 타입에 넣어 준다. 파싱을 소개하면서 쉽게 이해하기 위해 앞에서와 같이 시각적으로 접근하였다. 지금까지의 내용을 이해하였다면 다음 내용인 데이터 타입에 하나의 문자를 넣는 작업을 쉽게 이해할 수 있을 것이다.

1.2 연산이 가능한 데이터 만들기

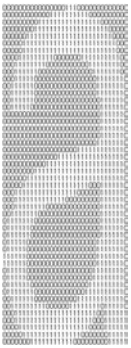
이미지 안의 문자는 단지 그림일 뿐이다. 프로그래밍하기 위해서는 문자 이미지를 데이터 형태로 가지고 있어야 한다. 우리는 파싱을 통하여 문자 이미지를 검은색과 흰색의 두 가지 색으로만 표현하였다. 여기에 더하여 문자의 크기도 일정하게 만들었다. 그렇다면 이것을 어떻게 새로운 데이터 타입 안에 담을 수 있을까?

컴퓨터에서 가장 작은 단위는 bit이며, 하나의 bit는 0과 1의 값으로 이루어져 있

다. 문자의 검은색을 1로 바꾸고 바탕의 흰색을 0으로 바꾼 후 bit를 사각형 안에 넣는다면 앞에서 파싱한 문자 이미지를 표현할 수 있다. 즉, [그림 1-5]처럼 'a'라는 이미지를 1,536개의 bit로 표현하는 것이 가능하다(가로 32개×세로 48개 = 1,536개).

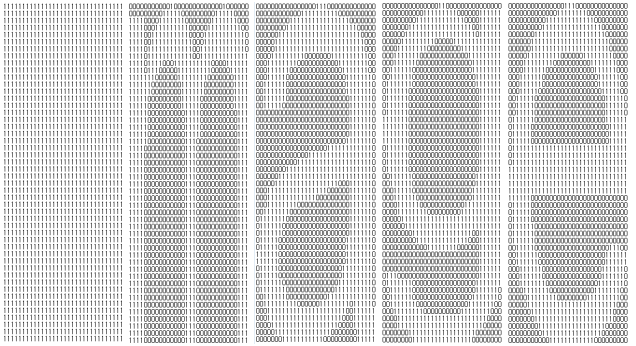
[그림 1-5]를 보면 'a'라는 것을 쉽게 알 수 있다. 0과 1만을 가지고도 시각적으로 쉽게 어떤 문자인지를 알 수 있다.

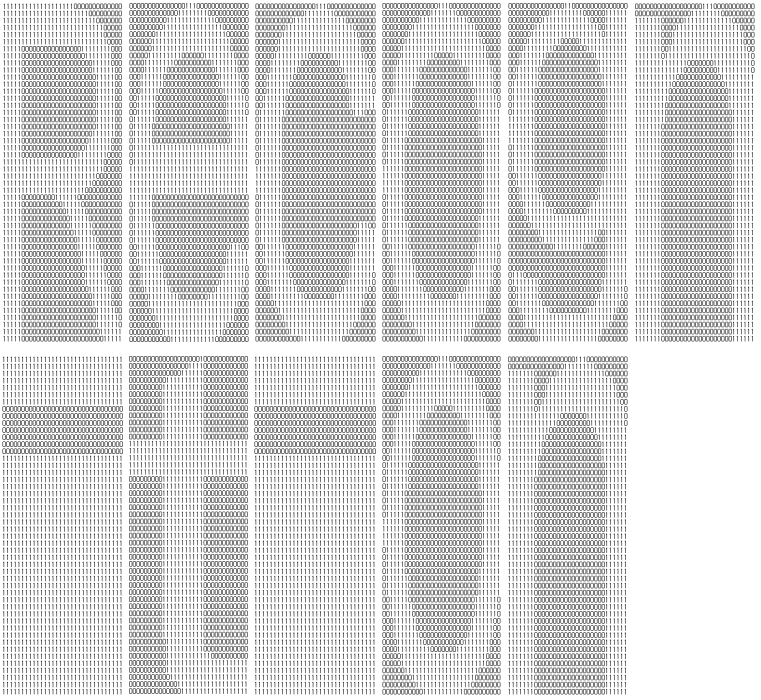
그림 1-5 0과 1로 표현된 문자 'a'



이와 같은 방법으로 'Image Recognition'이란 단어를 변환한다면 [그림 1-6]과 같이 나타날 것이다.

그림 1-6 0과 1로 표현된 'Image Recognition'





이미지의 크기를 32×48의 크기 안에 담은 것은 이유가 있는데, 프로그램 언어에서 정수형Integer 데이터 타입의 크기가 32bits이기 때문이다. 즉, 48개의 정수Integer를 하나의 새로운 데이터 타입으로 만들면 우리는 한 문자를 담는 그릇을 만든 것이다. 연산이 가능한 데이터 타입을 만드는 것은 프로그래머의 선택이며, 데이터 타입의 크기를 크게도 작게도 만들 수 있다.

이 책에서 다루게 될 데이터 타입은 두 가지다. 하나는 정수에 기반한 데이터 타입이며, 다른 하나는 8×12bits의 크기를 위한 문자형Character 데이터 타입에 기반한 데이터 타입이다. 앞으로 작은 문자 이미지에는 작은 그릇을 사용할 것이고, 큰 문자 이미지에는 큰 그릇을 사용할 것이다. 작은 이미지를 큰 데이터 타입에 넣는 것은 무의미하고 연산을 위한 속도도 더 느려지기 때문이다.

지금까지의 내용이 이미지 인식의 핵심이라 할 수 있다. 어느 정도 프로그래밍을

하는 독자는 이미지 인식에 어떻게 접근해야 하는지 이해하였을 것이다. 이해가 안 되었어도 앞으로 다루는 프로그래밍을 통하여 이미지 인식에 대하여 조금 더 깊이 있게 이해할 수 있을 것이다.

1.3 확률을 이용한 인식

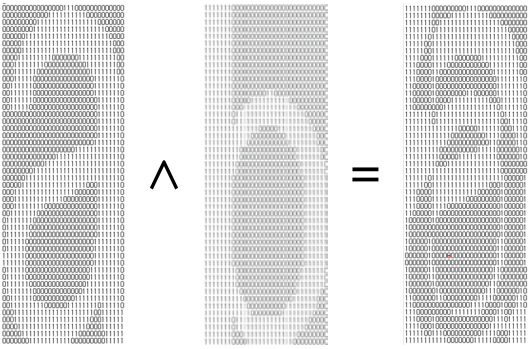
이미지 인식은 비교할 대상이 있어야 한다. 여기서는 비교를 위해 만들어진 이미지를 기본 이미지Standard Image라고 부르겠다. 이 책에서는 영문과 숫자만 인식하는데, 총 62개의 기본 이미지를 만들어서 사용한다(영문 소문자 26개+대문자 26개+숫자 10개). 인식할 이미지는 62개의 기본 이미지와 차례대로 비교하며 가장 근접한 이미지를 채택하여 결과값으로 결정한다. 그렇다면 연산할 수 있게 만든 데이터로 어떻게 bit가 같은지 비교하는지 알아보자.

프로그래밍 언어의 연산자 중 친근하지는 않지만 암호 프로그래밍 등을 하면 가장 많이 사용하는 것이 XOR(Exclusive OR, ^) 연산자다. 이 책에서도 XOR 연산자를 많이 사용할 것이다. XOR 연산자의 연산 결과는 다음과 같은데, 연산되는 bit가 같으면 0을, 다르면 1을 결과로 나타낸다.

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

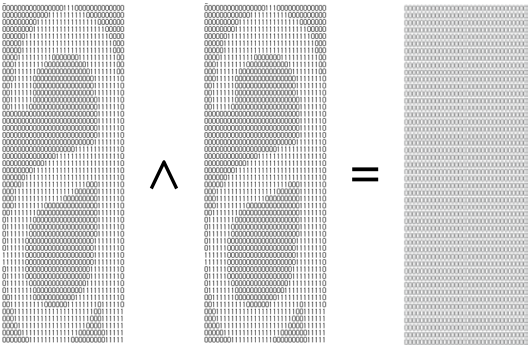
그렇다면 우리가 앞에서 연산할 수 있게 만든 데이터 타입과 어떻게 비교하면 될까? 기본 이미지 데이터Standard Image Data와 비교하기 위한 이미지 데이터Image Data를 위치가 같은 bit끼리 XOR 연산하면 된다. [그림 1-7]은 'a'와 'b' 두 개의 이미지 데이터를 비교한 것이다.

그림 1-7 a ^ b



서로 다른 두 개의 이미지를 비교하면 0과 1의 불특정 이미지 데이터로 나타난다. 하지만 두 개의 이미지가 동일하다면 XOR 연산자로 나타나는 이미지는 모두 0으로만 나타난다. [그림 1-8]은 'a'와 'a'를 XOR 연산자로 나타낸 것인데, 모두 0으로 나타나게 된다.

그림 1-8 a ^ a

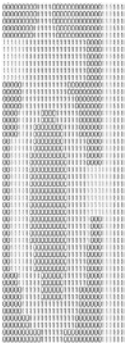


확률을 이용한 인식은 어떻게 이루어질까? 동일한 bit는 0으로 표현되기에 XOR 연산의 결과값을 가지고, 1,536개의 bit 중에 0의 숫자가 많은 것을 결과값으로 나타내면 된다(실제 이미지 인식에서는 모두 0으로 나오지 않는다). 하지만 연산을 통하여 0이 가장 많이 나온 기본 데이터를 결과로 출력하면 된다. 인식률이 높다는 것은 확률을 이용한 결과값이 올바른 결과값을 많이 도출한다는 것이다.

1.4 영문이 한글보다 인식률이 높은 이유

이 책에서 다루는 문자 인식은 영문과 숫자만을 대상으로 한다. 영문 인식은 62개의 기본 데이터를 만들면 되지만, 한글 인식은 만 개 이상의 기본 데이터를 만들어야 한다. 그런데 만 개 이상의 기본 데이터를 만들기도 어렵고 인식률도 떨어진다. 그렇다면 왜 한글 인식률이 높지 않은지 이미지 데이터를 가지고 알아보자.

그림 1-9 한글 이미지 데이터(허)



이 책에서 다루는 이미지 데이터는 가로 32개와 세로 48개 bit를 사용하는데, 이것으로 표현할 수 있는 총 가짓수는 1,536개다(가로 32개×세로 48개=1,536개). 이 이미지 데이터로 만 개가 넘는 한글을 표현하는 것은 불가능하다. 물론 우리는 더 큰 이미지 데이터를 만들어 사용할 수 있다. 하지만 만 개의 한글을 위한 기본 이미지 데이터를 만드는 것도 상당히 어려운 작업이며, 한 문자를 인식하기 위해서 만 개의 데이터와 항상 비교해야 한다면 연산 속도가 매우 느려질 것이다. 또한, 정확히 써진 한글만을 인식하는 환경이 아닌 곳에서는 이미지 변형을 수행하여 인식률을 높여야 하는데, 이미지 변형으로 인하여 오히려 다른 문자로 인식할 확률이 커질 수 있다.

chapter 3

문서 이미지 OCR

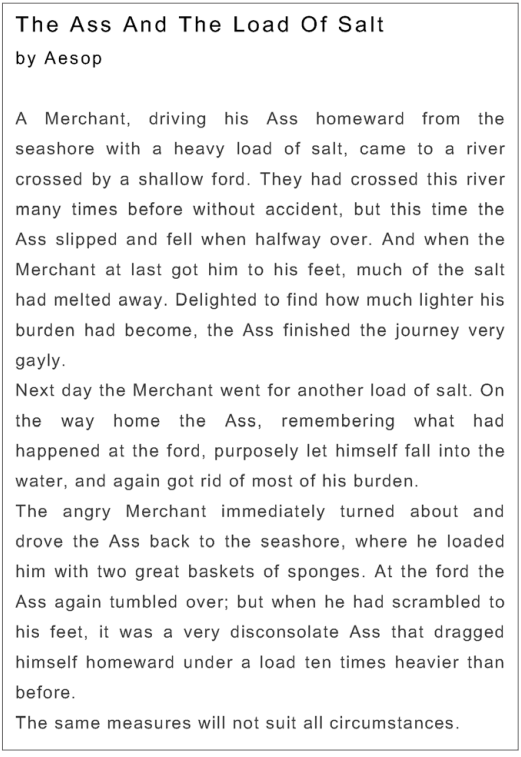
이번 장에서는 문서 이미지에서 문자를 판독해 내는 OCR 프로그램을 단계적으로 구현한다. 이전 장에서 기본적인 부분은 구현되었으나, OCR 프로그램의 다양한 부분에 대한 깊이 있는 분석과 추가적으로 구현해야 할 부분이 상당히 많다. 기본적인 내용으로 시작하지만, 내용이 진행됨에 따라 OCR에서 생각해야 할 부분을 조금 더 구체적으로 알 수 있으며, 이번 장이 끝나더라도 독자가 접하는 문서에 따라 추가적인 부분을 직접 구현할 수 있는 힘을 기르기를 바란다.

이 장에서 사용하는 문서 이미지는 '이솝 이야기'의 내용으로 [그림 3-1]과 [그림 3-2]의 두 개의 파일이다. 먼저 프로그램에서는 [그림 3-1]의 첫 번째 내용만을 가지고 OCR 프로그램이 구현되며, 모든 구현이 끝난 이후에는 [그림 3-2]의 내용도 OCR을 진행하여 프로그램의 완성도를 검증할 예정이다.

실생활에서도 문서 이미지 OCR을 많이 사용하고 있는데, 대표적으로 고급 복사기에서 많이 사용한다. 즉, OCR 기능을 가진 고급 복사기에서는 문서를 스캔하면 Word 파일로 결과가 만들어진다. 복사기는 문서를 스캔하여 문서 이미지를 얻으며, 복사기에 설치된 자체 프로그램이 문서 이미지를 판독하여 Word 문서 파일로 다시 만들어 준다. 이 기술이 발전하여 번역 프로그램과 결합할 경우를 생각해 보자. 책을 복사기로 복사하면 각 나라의 언어로 번역된 책이 자동적으로 만들어질 수도 있을 것이다. 아직까지는 번역 프로그램의 완성도가 아주 낮기 때문에 자동 번역에 대한 기능을 단시간에 만들기는 힘들다. 그러나 OCR가 자동 번역기와 함께하는 순간 우리 생활에 엄청난 변화를 가져다줄 것이다.

OCR 기술과 관련하여 생각해 볼 수 있는 또 한 가지는 문서를 읽어 주는 프로그램과의 결합이다. 복사기에서 책을 복사하면 성우가 책을 읽어 주는 소리 파일이 만들어진다. 이 기술은 OCR로 인식이 끝난 문서 파일에 인간의 음성으로 읽어 주는 기술을 접목하면 된다. 인간의 음성으로 읽어 주는 기술은 이미 시장에서 찾아볼 수 있으나 기술이 완벽하지는 않다. 이는 두 말이 이어져 발음되는 연음 처리에 대한 기술이 아직 완벽하지 않기 때문이다. 어쨌든, OCR 기술이 번역, 낭독 등의 기술과 함께할 때 아주 멋진 생활의 편리함을 가져올 것임은 분명하다.

그림 3-1 이솝 우화 문서 이미지(aesop_1.png: The Ass And The Load Of Salt)



The Ass And The Load Of Salt

by Aesop

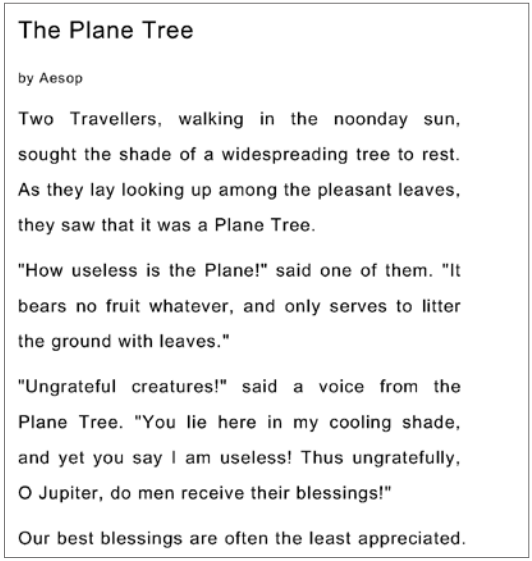
A Merchant, driving his Ass homeward from the seashore with a heavy load of salt, came to a river crossed by a shallow ford. They had crossed this river many times before without accident, but this time the Ass slipped and fell when halfway over. And when the Merchant at last got him to his feet, much of the salt had melted away. Delighted to find how much lighter his burden had become, the Ass finished the journey very gayly.

Next day the Merchant went for another load of salt. On the way home the Ass, remembering what had happened at the ford, purposely let himself fall into the water, and again got rid of most of his burden.

The angry Merchant immediately turned about and drove the Ass back to the seashore, where he loaded him with two great baskets of sponges. At the ford the Ass again tumbled over; but when he had scrambled to his feet, it was a very disconsolate Ass that dragged himself homeward under a load ten times heavier than before.

The same measures will not suit all circumstances.

그림 3-2 이솝 우화 문서 이미지 (aesop_2.png: The Plane Tree)



The Plane Tree

by Aesop

Two Travellers, walking in the noonday sun, sought the shade of a widespreading tree to rest. As they lay looking up among the pleasant leaves, they saw that it was a Plane Tree.

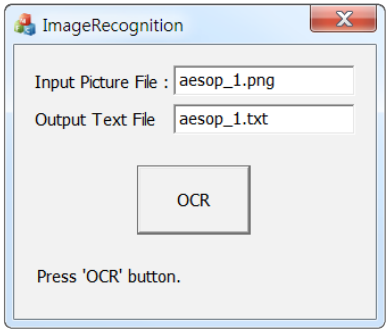
"How useless is the Plane!" said one of them. "It bears no fruit whatever, and only serves to litter the ground with leaves."

"Ungrateful creatures!" said a voice from the Plane Tree. "You lie here in my cooling shade, and yet you say I am useless! Thus ungratefully, O Jupiter, do men receive their blessings!"

Our best blessings are often the least appreciated.

[그림 3-3]은 이번 장에서 구현되는 프로그램의 실행 화면이다. 파일의 이름을 입력하고 'OCR' 버튼만 누르면 실행되는 프로그램으로, 상당히 간단하다. 'Input Picture File'에는 인식을 위한 문서 이미지 파일 이름을 넣고, 'Output Text File'에는 인식이 끝난 후의 결과 문자들을 입력하기 위한 파일 이름을 넣는다. 'OCR' 버튼을 누르게 되면 이후에 진행되는 모든 OCR 관련 프로그램 함수들이 실행된다.

그림 3-3 문서 이미지 OCR 프로그램 실행 화면



[코드 3-1]은 [그림 3-3]의 [OCR] 버튼을 눌렀을 때 실행되는 함수로, 'ImageRecognitionDlg.cpp' 파일에 있는 OnBnClickedBtnRunocr() 함수다.[01] ImageRecognitionDlg.cpp 파일에서는 버튼을 눌렀을 때 실행되는 [코드 3-1]의 내용이 가장 중요하며 OCR에서 중심이 되는 함수들은 'OCR.h'와 'OCR.cpp'에서 구현한다.

[코드 3-1] OCR 버튼을 눌렀을 때 실행되는 함수(ImageRecognitionDlg.cpp)

```
void CImageRecognitionDlg::OnBnClickedBtnRunocr()
{
    UpdateData(TRUE);

    m_message.Format(_T("OCR is Running..."));                    //---①
    UpdateData(FALSE);

    if (image != NULL)
        image.Destroy();

    HRESULT hResult = image.Load(m_inputfile);                    //---②

    if (FAILED(hResult)) {
        m_message.Format(_T("Error : Can't Open Input File."));
        UpdateData(FALSE);
        return;
    }

    ocr->RunOCR(&image, m_outputfile, 50);                        //---③

    m_message.Format(_T("OCR is Completed..."));                  //---④
    UpdateData(FALSE);
}
```

① 실행 화면 하단의 'Press 'OCR' button.' 메시지가 'OCR is Running…'으로 바뀐다. UpdateData(TRUE)는 화면의 내용을 프로그램의 각 멤버 변수 내용으로 업데이트하고, UpdateData(FALSE)는 멤버 변수의 내용이 화면의 내용을 바꾸기 때문에 m_message의 내용이 화면 하단의 메시지를 바꾸게 된다.

② 문서 이미지 파일의 이름을 저장하고 있는 m_inputfile 변수를 사용하여 이미지를 로

01 MFC 프로그램에서 버튼 등을 포함한 도구와 해당 도구와 연동되는 멤버 변수의 추가에 대해서는 『MFC 프로그래밍 : 주식 분석 프로그램 만들기』(한빛미디어, 2014)에서 구체적으로 설명하였으므로 여기서는 설명을 생략한다.

드한다. 즉, 문서 이미지가 CImage 클래스 변수인 image에 저장된다.

③ COCR 클래스의 멤버 함수인 RunOCR을 실행해 OCR을 진행한다. 이때 전달되는 파라미터는 이미지를 담고 있는 image, 출력 파일 이름인 m_outputfile, RGB에서 검은색(문자 색)을 나타내는 50이다.

④ OCR이 실행되면 화면 하단에 'OCR is Completed…' 메시지로 종료를 알려 준다.

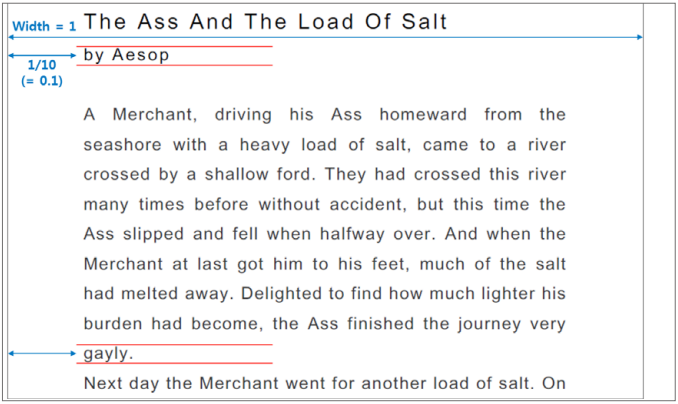
Dialog 파일에서의 구현은 이처럼 간단하다. 이후에는 확률을 이용한 OCR의 기본 개념부터 OCR에 추가로 구현해야 하는 내용을 단계적으로 알아본다.

3.1 확률을 이용한 OCR

이전까지 기본 이미지 데이터를 만드는 방법을 알아보았다. 이것은 사전이 만들어진 것으로 이해하면 된다. 지금부터 진행되는 OCR의 실행 단계는 하나의 문자를 이미지 파싱을 거쳐서 이미지 데이터를 만든 후, 사전과 같은 기본 이미지 데이터와 비교하여 특정 문자로 결정하는 단계다. 3장에서는 처음부터 단계적으로 OCR을 진행하게 되는데, 먼저 확률을 이용한 OCR이 어떻게 이루어지는지 알아볼 것이다. 이것이 OCR의 가장 기본이 되는 기술이다. 그러나 확률을 이용해서 문자를 결정하는 방법만으로는 인식의 한계를 알게 될 것이다. 기본 62개의 문자보다 더 많은 특수 문자가 있으며, 줄바꿈Enter과 공백Space 등의 부분도 구현해야 하기 때문이다.

[그림 3-4]는 이번 장에서 인식할 문서 이미지인데, 이미지 파싱을 어디서부터 해야 하는지를 보기 위한 그림이다. 문서 이미지에서는 이미지 파싱이 중간이 아닌 앞쪽에서 이루어져야 한다. 심한 경우에는 한 줄에서 앞쪽에 짧은 한 단어만이 존재할 수 있기 때문이다. 프로그램에서는 이 부분을 문서의 가로Width 크기에서 비율Rate로 비교하여 앞쪽에서 0.1부터 시작하여 0.3까지 픽셀을 검사하여 한 줄을 인식하는 것으로 구현하였다.

그림 3-4 이미지 파싱의 기준



[코드 3-2]는 COCR 클래스의 헤더header 파일로, 이미지 파싱을 위해 시작되고 끝나는 가로의 비율을 처음에 0.1과 0.3으로 정의하였다. 이후에 구현되는 함수를 추가로 클래스의 멤버 함수로 정의하였다. 이는 2장에서 사용한 내용과 거의 유사하며 확률을 이용하여 특정 문자로 인식하는 함수를 추가로 구현하였다.

[코드 3-2] COCR 클래스 멤버 함수(OCR.h)

```
#define RATE_START_FOR_PARSING    0.1                                    //---①
#define RATE_END_FOR_PARSING      0.3                                    //---②

##### 생략 #####

class COCR {

private:
    CImage *image;
    Standard standard;                                                   //---③
    AllData allData;
    Data *data;

    int colorToCheck;

public:
    COCR(void);
    ~COCR(void);

    void RunOCR(CImage *image, CString outFileName, int colorLetter);//---④
    void GetStandardImageDataFromBinaryFile(char * fileName);          //---⑤
```

```
    void ParsingStepFirst();
    void ParsingStepSecond(int yTop, int yBottom);
    void ParsingStepThird(Rect *rect);
    void ParsingStepThird2(Rect *rect);
    void MakeImageData();
    void FindLetterValue();                                    //---⑥
    void StoreLetterToTextFile(CString outFileName);           //---⑦
};
```

① RATE_START_FOR_PARSING을 0.1로 정의한다. 상수 변수의 이름에서 알 수 있듯이 이미지 파싱을 위한 시작 비율을 나타낸다([그림 3-3] 참조).

② RATE_END_FOR_PARSING은 0.3으로 정의하며 이미지 파싱을 위한 마지막 부분의 비율을 나타낸다.

③ 기본 이미지 데이터를 저장하는 standard 변수를 정의한다. 나머지 변수는 2장과 동일한 목적으로 사용되었다.

④ OCR을 실행하는 함수로, 매개변수로 이미지 클래스 변수인 image, 출력 파일의 이름, 문자로 인식하기 위한 색의 값을 가진다.

⑤ 프로그램이 실행되면 자동으로 COCR 클래스가 생성되고, 생성 클래스에서 자동으로 GetStandardImageDataFromBinaryFile( ) 함수를 실행한다. 즉, 프로그램이 실행될 때 바이너리 파일로부터 기본 이미지 데이터를 읽어서 Standard 구조체에 넣는다.

⑥ 확률을 이용하여 Standard 구조체에 저장된 값과 비교하여 인식하려는 문자의 결과값을 얻게 된다.

⑦ 인식된 이미지의 결과들을 outFileName의 파일 이름에 텍스트 형태로 저장한다.

[코드 3-3]은 COCR의 생성자와 프로그램 실행 화면에서 [OCR] 버튼을 선택할 때 실행되는 함수를 구현한 것이다. 인식하려는 문서에서 이미지 파싱을 진행한 후 각각의 문자를 이미지 데이터로 만든다. 그다음 기본 이미지 데이터와 비교하여 특정 문자로 인식하고 결과 파일에 저장한다.

[코드 3-3] COCR 생성자 및 OCR 실행 함수 RunOCR(OCR.cpp)

```
COCR::COCR(void)
{
    GetStandardImageDataFromBinaryFile("standard.bin");    //---①
}
```

```
COCR::~COCR(void)
{

}

void COCR::RunOCR(CImage *newImage, CString outFileName, int colorLetter) {

    image = newImage;                                    //---②
    colorToCheck = colorLetter;                          //---③

    for (int i=0; i<MAX_COUNT_DATA; i++)
        allData.data[i].isFixed = false;                 //---④

    ParsingStepFirst();                                  //---⑤
    MakeImageData();                                     //---⑥

    FindLetterValue();                                   //---⑦

    StoreLetterToTextFile(outFileName);                  //---⑧
}
```

① 프로그램이 실행될 때 COCR 클래스의 생성자가 실행된다. 바이너리 파일에서 기본 이미지 데이터를 읽어서 기본 데이터 구조체를 만든다.

② 포인터 변수 image로 인식하려는 이미지를 가리킨다. 이후에는 image 변수로 해당 이미지에 접근하게 된다.

③ 문자 색을 colorToCheck의 변수에 저장한다.

④ allData는 인식하기 위해 파싱된 모든 문자를 저장한다. 아직 문자가 정의되지 않았기 때문에 isFixed 변수를 false로 초기화한다. 이후에 문자가 확률을 통하여 특정 문자로 결정되면 isFixed 변수는 true가 된다.

⑤ 이미지 파싱을 진행하는 함수로, 2장에서 구현한 것과 동일하다. 단지, 2장에서는 문자선 가운데를 기준으로 나누었다면, 이번에는 앞쪽을 기준으로 나누었다는 차이가 있다. ParingStepFirst( ) 함수의 설명은 2장에서 했으므로 여기서는 생략한다.

⑥ 이미지 파싱된 문자 이미지를 이미지 데이터에 넣는 함수로, 이 함수도 2장에서 구현한 것과 동일하므로 여기서는 설명을 생략한다.

⑦ 기본 이미지 데이터와 비교하여 확률이 가장 높은 문자를 인식하려는 문자의 결과값으로 정의한다.

⑧ OCR의 인식이 끝난 상태이므로 결과값을 텍스트 파일에 저장한다.