

EECS 22L: Project 2

Prepared by: Che-Wei Chang, Yasaman Samei, and Prof. Rainer Dömer

Feb 11, 2014

1 Optical Character Recognition (OCR)

This is the second team programming project of EECS 22L, "Software Engineering Project in C language".

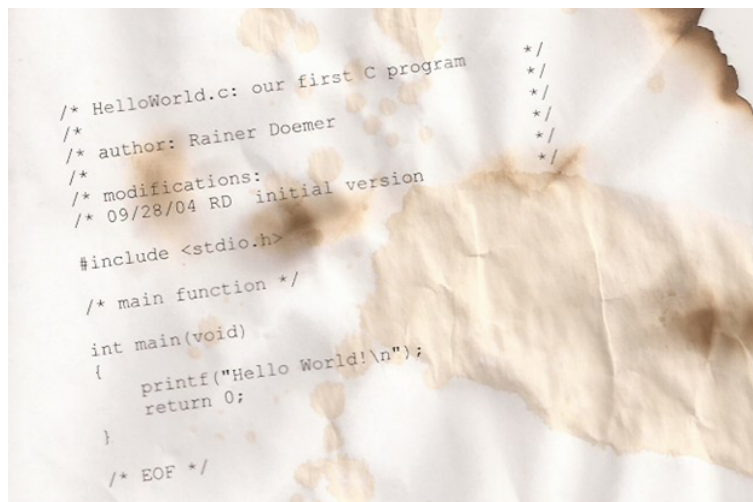
The goal of this programming exercise is to develop an OCR (Optical Character Recognition) program with which a user can convert scanned images of printed text into a digitized text file. OCR is a common method of digitizing printed texts so that these texts can be stored or processed in a more efficient way.

This project is designed to be an interesting exercise where you can practice all elements of software programming and engineering and work in teams. In particular, you will practice specifying and documenting the software program, designing data structures and algorithms, designing software modules, writing original source code, testing and debugging the software program, and collaborating and communicating effectively in a team.

1.1 Motivating Story

Mr. Nob Ackup runs a small software company that implements small-scale projects for its clients. One day, a thunder storm strikes. Lightning creates a sudden power surge followed by an outage. After power is restored, Mr. Nob Ackup finds his computer and backup systems damaged and can only partially recover the source code of his projects from old tapes. Luckily, his software engineers find stacks of papers with printed hard copies of most missing source files that can be scanned into digital images.

The task of your project team is to design a software package for Mr. Nob Ackup's company so that he can efficiently convert the scanned images of printed source code back into compilable files.



```
/* HelloWorld.c: our first C program */
/* author: Rainer Doemer */
/* modifications: */
/* 09/28/04 RD initial version */

#include <stdio.h>

/* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}

/* EOF */
```

Figure 1: Example of the "recovered" source code

2 Program Specification

In this project, we want a software program to be built, with which the user can read a scanned image, interactively process the image and convert it back into compilable source code files.

There are several features that we expect the OCR program to have. We distinguish between the minimum requirements for this project, and more advanced options that are goals and optional features (bonus points).

2.1 Basic functions that are required:

1. Graphical user interface (GUI) for the interactive OCR program
2. Loading a scanned image into the program and displaying it
 - (a) image in JPEG format
3. Image preprocessing:
 - (a) black and white conversion
 - (b) stain and wrinkle removal
 - (c) rotation (to level a slanted scan)
 - (d) cropping (to select sections suitable for OCR)
4. OCR:
 - (a) identifying and cropping a sequence of single characters from the image
 - (b) comparing cropped characters with reference character images from a font data base (we supply scanned font "Courier New", 12 point, at 300 DPI)
 - (c) storing recognized characters in a text file data structure
5. Text postprocessing
 - (a) display the recognized text
6. Text file output
 - (a) store the recognized text in a text file
7. Automated unit test for all major program modules

2.2 Advanced options that are desirable (but optional): (Bonus)

In addition to the above minimum requirements, additional features may be inserted:

- 2. (b) Read multiple input image formats, such as .ppm, pbm, or .bmp, in addition to .jpg.
- 2. (c) Combine multiple input images (scanned pages) into a single output text file
- 4. (d) In addition to the supplied "Courier New" font, support other fonts (e.g. Helvetica)
- 4. (e) In addition to the regular mono-space style, support other styles (e.g. italic or bold)
- 4. (f) In addition to 12 point scanned at 300 DPI, support other sizes (e.g. 10pt at 200 DPI)
- 5. (b) allow a text editor to be used on the recognized text (for the user to apply edits)
- 5. (c) Dictionary support: since it is very difficult for OCR to distinguish similar-looking characters (e.g. l and 1, 0 and O, C and G), a dictionary may be used to fix such "typos"

- 6. (b) Store the recognized text in other formats
- 6. (c) Run the C compiler on the recognized source code

Of course, any other options are also welcome that improve the OCR program towards better character recognition, faster processing, or easier use.

3 Software Engineering Approach

In the design and implementation of this project, we will follow basic principles of software engineering in a close-to-real-world setting. You will practice the major tasks in software engineering to build your own software product. As for project 1, we will not provide detailed instructions on how to design the program. Instead, your team needs to come up with your own choices and practice designing the software architecture of a medium-size program and document it.

3.1 Team work

The software design and programming in this course will be performed by student teams. Teams of 6-8 students will be formed at the beginning of this project. Team work is an essential aspect of this class and every student needs to contribute to the team effort. While tasks may be assigned in a team to individual members, all members eventually share the responsibility for the project deliverables.

The overall tasks of software design, implementation and documentation should be partitioned among the team members, for example, to be performed by individuals or pairs (pair programming). A possible separation into tasks or program modules may include:

- main program
- graphical user interface
- image processing (data structures and basic functions for images)
- reference font data base (lists of reference images)
- preprocessing (binarization, stain/wrinkle removal, cropping, image rotation)
- character cropping
- character matching (similarity comparison)
- postprocessing (e.g. matching most likely word in a dictionary)
- documentation
- testing

When planning the team partitioning, keep in mind that certain tasks depend on others and that some tasks are best handled by everyone together. However, for this OCR project in particular, there is a good opportunity to exploit the fact that the overall control flow of the program is **pipelined**. The stages of the pipeline can easily be implemented and tested separately. A simple file interface between the stages of preprocessing, OCR, and postprocessing can make these modules (and even their sub-modules) independent, so that they can be addressed separately and in parallel.

A team account will be provided on the *crystalcove* and *zuma* server for each team to share source codes, data and document files among the team members. Since teams will compete in the projects, sharing of files across teams is not permitted.

Every student is expected to show up and participate in team meetings. Attendance of the weekly discussion and lab sections is mandatory for the sake of successful team work.

3.2 Major project tasks

As for the previous project, we again go through several steps to approach this medium-sized programming project, with **some changes**.

- *Design the software application specification:* work as a team to decide the functionalities of the program, the *input* and *output* of the program, and other things that describe the *features* of the program for the users. Again, we will write an user manual to document this.
- *Design the software architecture specification:* work as a team to design the data structure, program modules, application programming interface (API) functions between modules, and basic algorithms that will be used to solve the problem. This will be documented again in a software specification document, but special attention should be given to the **testing plan**.
- *Build the software package:* write the source code and implement the program. Each team member may be in charge of their own module(s) and ideally work in parallel on implementation and module testing. Use Makefile for rule-based compilation to integrate the modules from different owners. Here, will use **unit testing** for each module before integration into the entire software system.
- *Version control and collaboration:* use a version control application, i.e. CVS (introduced in Lecture 3) to maintain the team project documentation and source code files. Team members can synchronize their own work with the others through the team repository located in the team account. In contrast to Project 1, **the use of CVS in Project 2 is mandatory**.
- *Test and debug the software:* work as a team to decide the testing strategies, write automated test programs or scripts, and debug the program when some of the test cases fail. Both unit tests and **system test** are needed during software development and delivery.
- *Software release:* release the software package with the executable program and documentation, e.g. the README file, user manual, etc. Release also the source code as a package for the further developers or maintainers. In contrast to Project 1, Project 2 has 3 software releases, **alpha, beta, and final release**.

3.3 Deliverables

Each team needs to work together and submit one set of deliverables each week. Here is the checklist of the files the team needs to submit and the due dates (hard deadlines).

Table 1: The OCR Project Deliverables

Week	File Name	File Description	Due Date
1	OCR_UserManual.pdf	The application specification	02/17/14 at 12:00pm
2	OCR_SoftwareSpec.pdf	The software architecture specification	02/24/14 at 12:00pm
3	OCR_Alpha.tar.gz OCR_Alpha_src.tar.gz	The alpha version of the OCR program, including the program source code and documentation	03/03/14 at 12:00pm
4	OCR_Beta.tar.gz OCR_Beta_src.tar.gz	The beta version of the OCR program, including the program source code and documentation	03/10/14 at 12:00pm
5	OCR_V1.0.tar.gz OCR_V1.0_src.tar.gz	The release software package for the OCR program and the program source code and documentation	03/17/14 at 12:00pm

Note that we do require these exact file names. If you use different file names, we will not see your files for grading.

We will separately provide detailed templates (document skeleton, table of expected contents) for the textual documents and a detailed list of contents (directory structure and expected files) for the file archives. These grading criteria will be provided at the Projects tab of the course webpage.

3.4 Submission for grading

To submit your team's work, you have to be logged in the server `zuma` or `crystalcove` by using your **team's account**. Also, you need to create a directory named `ocr` in your team account, and put all the deliverables in that directory. Next, change the current directory to the directory containing the `ocr` directory. Then type the command:

```
% /ecelib/bin/turnin22
```

which will guide you through the submission process.

For each deliverable, You will be asked if you want to submit the script file. Type yes or no. If you type "n" or "y" or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.

Below is an example of how you would submit your team work:

```
zuma% ls # This step is just to make sure that you are in the correct directory that contains ocr/
ocr/
zuma% /ecelib/bin/turnin22
=====
EECSL 22L Winter 2014:
Project "OCR" submission for team1
Due date: Mon Feb 17 12:00:00 2014
* Looking for files:
* OCR.UserManual.pdf
=====
Please confirm the following: *
"I have read the Section on Academic Honesty in the *
UCI Catalogue of Classes (available online at *
http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) *
and submit original work accordingly." *
Please type YES to confirm. y
=====
Submit OCR.UserManual.pdf [yes, no]? y
File OCR.UserManual.pdf has been submitted
=====
Summary:
=====
Submitted on Sun Feb 2 00:55:31 2014
You just submitted file(s):
OCR.UserManual.pdf
zuma% _
```

You may want to verify your submission. This step is optional, but recommended. If you want to confirm which files you have submitted, call the following command:

```
zuma% /users/grad2/doemer/eecs22/bin/listfiles.py
```

This command lists your submitted files. More importantly, be sure to double-check that the submitted packages can be cleanly extracted and used (graded!).

For a binary package, we expect the user to read the documentation and run the executable program as follows:

```
% gtar xvzf BinaryArchive.tar.gz
% evince ocr/doc/OCR.UserManual.pdf
% ocr/bin/ocr
```

For a source code package, we expect the developer to read the documentation and build the software as follows:

```

% gtar xvzf SourceArchive.tar.gz
% evince ocr/doc/OCR_SoftwareSpec.pdf
% cd ocr
% make
% make test
% make clean

```

Again, please ensure that these commands execute cleanly on your submitted packages.

4 Design and Implementation Hints

For this second project in this course, we would like to provide a few suggestions towards effective software design and implementation.

4.1 A overview of an OCR program

In this section, we will show an architecture of a simple OCR program. You can divide your program into multiple modules according to this architecture.

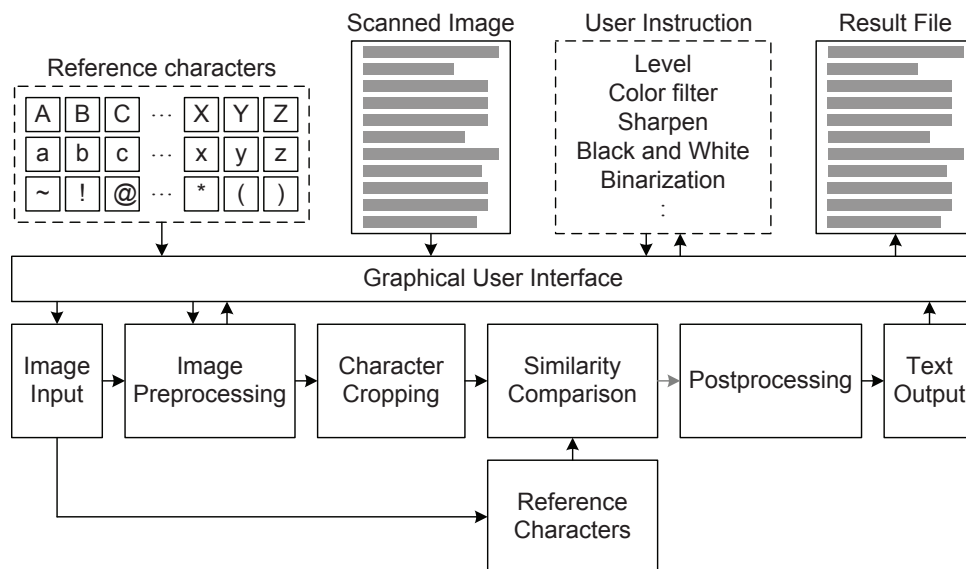


Figure 2: Pipelined modules in an OCR program

In this illustration, the OCR program is divided into multiple modules, including file I/O, image preprocessing, character cropping, similarity comparison and post-processing. Each module can be and should be constructed and tested independently.

4.2 An object-oriented data structure

While the ANSI-C programming language is not explicitly an object-oriented language (that would be C++), it is nevertheless prudent to think of the objects at hand when designing a data structure for a project.

For the OCR program, the following are some of the basic objects that require representation in the program's data structures:

- an image (a scanned image and/or a reference character)
- a list of images (a list of cropped and reference characters)

- a list of output words (a list of recognized characters)
- a list of keywords (a list of ANSI-C keywords, useful when processing C code)

Together with such object data, methods or functions that manipulate the objects are also needed. For the OCR program, suitable basic functions include the following:

- specify the input and output file name(s)
- process an input image, and show the processed image in the graphical user interface
- locate a character in an input image
- compare a character image with a reference character
- etc.

Additionally, specific higher-level functions are needed for various purposes, including:

- read reference characters and scanned image into the data structure
- apply image processing to the scanned image
- locate and crop characters from the scanned image
- determine the best matching reference character for a given cropped character
- search a keyword in the dictionary (optional)
- output the resulting text to a file
- etc.

4.3 Overall OCR program control flow

The basic overall control flow of a OCR program is the following:

- setup (read reference character library)
- read the scanned image
- display the scanned image
- repeat
 - get image processing instruction from user
 - apply image processing to the image
 - display the processed image
- repeat
 - locate and crop a character from the image
 - match the character with the reference characters
 - put recognized characters into the text data structure
- keyword matching (optional)
- output the recognized text to a file

4.4 Unit test for all major modules

In this project, unit test for all major modules is required. The OCR program can be divided into at least four major modules, which are **GUI**, **preprocessing**, **OCR** (including **cropping** and **matching** process) and **postprocessing**. These four modules can be, and should be constructed and tested independently. In the software deliverable, you have to provide test functions for these four modules. Each test function should contain a main function in which the module is individually tested, e.g. with hard-coded input. The test function should also display the result on the screen so that the designer can verify if the module works correctly or not.

Figure 3 shows an example of unit test for the cropping function inside the OCR module.

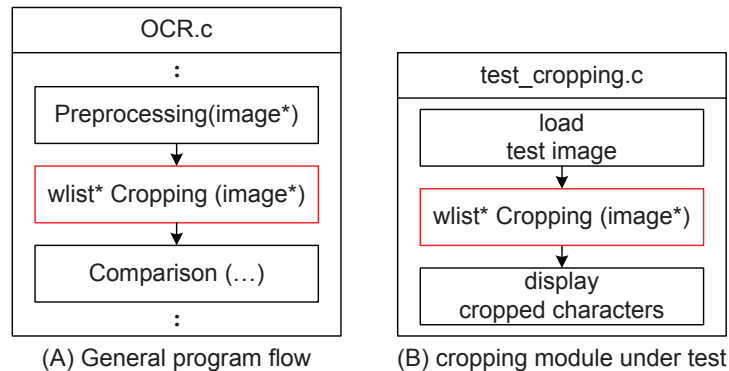


Figure 3: Unit test of Cropping function in the OCR program

Also, your top-level Makefile in the software deliverable should provide options to run the test function. We expect the developer to run the unit tests as follows:

```
% make test_gui
% make test_preproc
% make test_ocr
% make test_postproc
```

4.5 Sample Images

For this project, since we do not have scanning hardware available for each team, we will supply a set of example images of scanned source code, as well as a library of reference character images. Initially, the following image files will be available in the eecs22 account:

```
~eecs22/OCR/01_HelloWorld.Clean300DPI.jpg - clean scan of HelloWorld.c
~eecs22/OCR/02_HelloWorld.Wrinkles300DPI.jpg - wrinkled version of the previous
~eecs22/OCR/03_HelloWorld.Coffee300DPI.jpg - coffee-stained version of the previous
~eecs22/OCR/04_HelloWorld.Burned300DPI.jpg - partially burned version of the previous
~eecs22/OCR/05_HelloWorld.Slanted300DPI.jpg - slanted scan of the previous
~eecs22/OCR/CourierNew12.docx - original document with Courier New font, 12 point
~eecs22/OCR/CourierNew12.pdf - PDF version of the previous
~eecs22/OCR/CourierNew12_300DPI.jpg - clean scan of Courier New font sample
~eecs22/OCR/CourierNew12_300DPI.ppm - portable pixel map (PPM) format of the previous
~eecs22/OCR/CourierNew12_300DPI.pbm - black&white version of the previous
~eecs22/OCR/CourierNew12_300DPI/33.jpg - 30x46 pixel image of ASCII code 33 (!)
~eecs22/OCR/CourierNew12_300DPI/... .jpg ...
~eecs22/OCR/CourierNew12_300DPI/126.jpg - 30x46 pixel image of ASCII code 126 (~)
```

Note that all of the *HelloWorld* images may be used as input to the OCR program (with increasing difficulty).

The images in the CourierNew12.300DPI directory can serve as starting point for your database of character images. Note that these images have been cut from the scanned font page and stored under the name of their decimal ASCII value.

Additional scanned images may be added throughout the project.

4.6 Image processing

Naturally, OCR relies on a number of basic and advanced image processing operations. Here, your experience with the image processing assignments in the prior course EECS 22 "Advanced C Programming in C Language" will come in handy. Feel free to reuse and extend any of the source code of the PhotoLab application for this project.

In addition, you may make use of a third-party software package available on the department Linux servers, namely the NetPBM set of packages. NetPBM is a rich collection of image processing tools and library functions for reading, writing, converting, and manipulating NetPBM images.

The manual pages for NetPBM and libnetpbm are probably a good starting point:

```
firefox /usr/share/doc/netpbm-10.47.05/userguide/index.html  
firefox /usr/share/doc/netpbm-10.47.05/userguide/libnetpbm.html
```

While some of the image processing operations are available directly in the `libnetpbm` via a proper C application programming interface (API), other image manipulation operations are only available as command line tools, e.g. `pnmrotate`. For the latter, you can run these from within your C program by use of the NetPBM `pm_system()` or the general Linux `system()` call.