보고서

영상 회전 프로그램

전자정보통신공학과 23010860 이윤지

보고서

목차

과제 목표

과제 목표

파일 입출력

프로그램이 외부 파일과
효율적으로 상호작용할 수 있도록 함.

동적 메모리 할당

실행 시 필요한 만큼만 메모리를
사용할 수 있어 효율적임.

이미지 처리

영상(이미지)의 기본 구조와
파일 형식에 대해 이해할 수 있음.

포인터

메모리를 직접 조작하여
효율적인 프로그래밍을 가능하게 함.

코드 작성

실제 응용 능력을 향상 및
영상 처리에 대한 이해를 넓힘.

과제 목표

첫 번째 “영상 회전 프로그램 제작” 과제의 목표는 다음과 같다.

단순히 코드 제작에만 집중하는 것이 아닌, 이후 계속 사용해야하는 C언어 개념에 대한 학습의 기회로 삼고자 한다.

특히 이 프로그램 제작에 주로 사용되는 “포인터”와 “동적 메모리 할당”, "파일 입출력" 등의 핵심 개념에 대한 지식을 학습하고자 한다.

더 나아가 영상 처리의 기본이 되는 디지털 영상에 대한 기본적인 정보를 정리하여, 프로그램 개발에 필요한 이론적 배경을 다지고자 한다.

또한 이 보고서에서는 C언어를 사용하여 영상을 회전시키는 프로그램의 개발 과정을 다룰 것이다.

프로그램 구현에 필요한 알고리즘, 주요 함수 설계 및 구현 방법에 대해 설명할 예정이다.

이를 통해 C언어의 실제 응용 능력을 향상시키고, 영상 처리 분야에 대한 이해를 넓히는 것이 이 과제의 궁극적인 목표이다.

C언어 기본 개념

1. 포인터

포인터는 변수의 메모리 주소를 저장하는 변수이다. C언어에서 포인터는 메모리를 직접 조작하고 효율적인 프로그래밍을 가능하게 한다.

선언 방법	**자료형 *변수명;**	
초기화 방법	**p = &i;**	(변수 i의 주소를 포인터 p에 저장할 때)
참조 방법	***p**	(포인터 p가 가리키는 메모리 위치의 값 반환)

&: 주소 연산자, *: 간접 참조 연산자

포인터를 사용할 때 주의할 점은 초기화되지 않은 포인터를 사용하면 예측할 수 없는 결과가 발생할 수 있기 때문에, 반드시 초기화해야 한다.

또한 C에서 배열 이름은 배열의 첫 번째 요소의 주소를 나타내는 포인터 상수로 취급되기에 포인터로 배열의 요소에 접근할 수도 있고,
포인터는 인덱스를 주소로 변환할 필요가 없기 때문에 인덱스 표기법을 사용하는 배열보다 빠른 장점이 있다.

2. 동적 메모리 할당

동적 메모리 할당은 프로그램 실행 중에 필요한 만큼의 메모리를 할당받고 사용이 끝나면 시스템에 반환하는 메모리 관리 방식이다.
이는 프로그램 시작 전에 메모리 크기가 결정되어 유연성이 떨어지는 정적 메모리 할당의 한계를 극복하기 위해 사용된다.
동적으로 할당된 메모리는 포인터를 통해 접근할 수 있다. 배열처럼 인덱싱하거나 포인터 연산을 사용할 수 있다.

malloc	**지정된 바이트 수만큼의 메모리를 할당**
calloc	**malloc()과 유사, 할당된 메모리를 0으로 초기화**
realloc	**이미 할당된 메모리의 크기를 변경**

ex. int *p = (int *)malloc(100 * sizeof(int)); [malloc은 void를 반환하므로, 적절한 자료형으로 형변환 필요]

메모리 누수 방지를 위해, 동적으로 할당된 메모리는 사용이 끝나면 반드시 free() 함수를 사용하여 시스템에 반환해야 한다.

* 이미 해제된 메모리를 다시 해제하거나 사용하는 것은 위험하므로 주의

[참고자료] "정민채, '프로그래밍 강의 내 포인터, 동적 메모리 할당 자료', 2023년 2학기"

3. 파일 입출력

C언어에서의 파일 입출력은 이미지 처리 프로그램에서 중요한 역할을 한다.

파일 작업의 핵심은 FILE 구조체에 대한 포인터인 파일 포인터인데, 이는 다음과 같이 선언한다:

선언 방법

```
File *file_pointer;
```

파일을 열고 닫는 기본 함수는 fopen()과 fclose()이다:

```
file_pointer = fopen("filename.txt", "mode");
```

```
fclose(file_pointer);
```

여기서 "mode"는 파일의 사용 목적을 정의한다. 주요 모드로는 읽기("r"), 쓰기("w"), 추가("a")가 있다.
쓰기 모드는 기존 파일 내용을 지우고 새로 작성하며, 추가 모드는 기존 파일 끝에 데이터를 추가한다.
파일 모드는 텍스트 모드와 이진 모드로 나뉜다. 텍스트 모드("rt", "wt", "at")는 줄바꿈 문자 등을 시스템에 맞게 자동 변환하지만,
이진 모드("rb", "wb", "ab")는 데이터를 변환 없이 그대로 처리한다.

* 이 과제에서는 RAW 이미지 파일을 다루므로 이진 모드를 사용한다. 이는 이미지 데이터의 무결성을 보장하고 원본 정보를 그대로 유지하기 위함이다.

[참고자료] "https://ko.wikipedia.org/wiki/C_%ED%8C%8C%EC%9D%BC_%EC%9E%85%EC%B6%9C%EB%A0%A5"

영상(이미지) 처리

1. 디지털 이미지의 기본 구조

디지털 이미지의 기본 구성 요소는 아래와 같다:

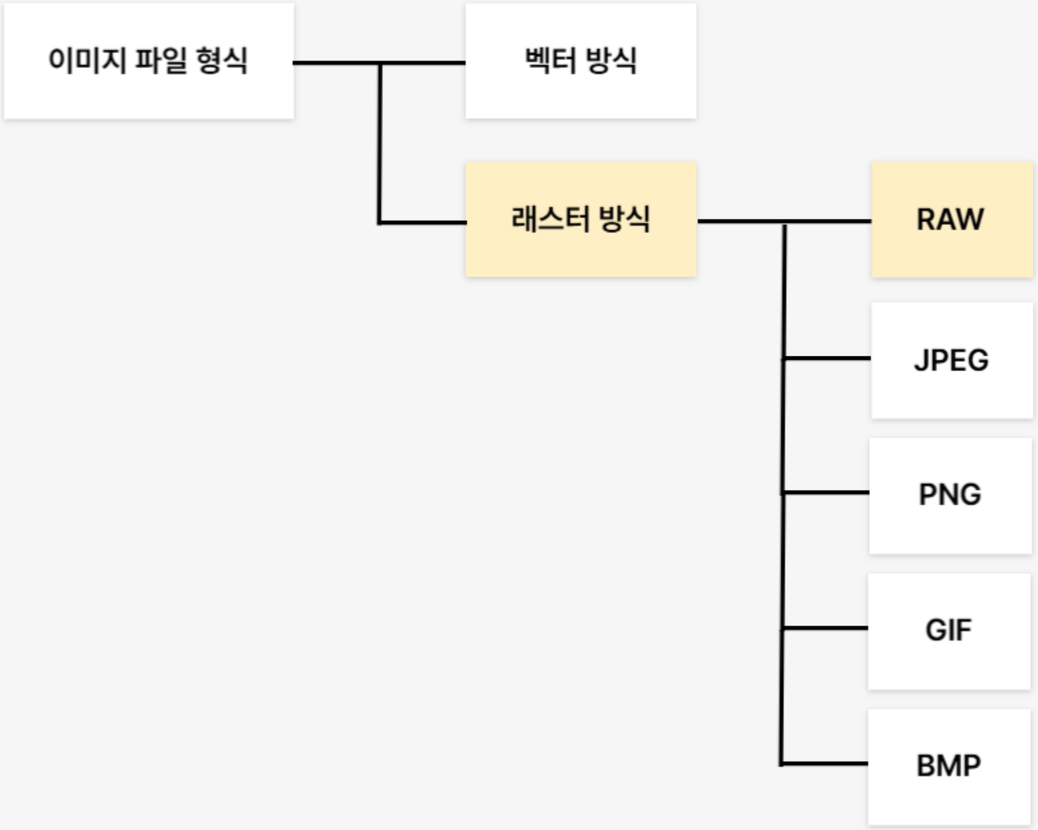
픽셀 — 디지털 화면을 구성하는 가장 작은 단위로, 정사각형 형태의 색상 점

해상도 — 단위 면적당 포함된 픽셀의 개수를 나타내며, 이미지의 선명도를 결정

* 높은 해상도는 더 많은 픽셀을 포함하므로 더 선명하고 세밀한 이미지를 제공하게 된다.

[참고자료] "길벗알앤디, "시나공 총정리 컴퓨터활용능력 1급 필기", 2020"

2. 이미지 파일 형식



이번 과제에서 사용되는 래스터 방식의 형식에 대해서만 간단히 알아보고자 한다.

래스터 방식은 픽셀로 구성된 이미지를 저장하는 형식이다.
JPEG는 손실 압축 방식을 사용해 파일 크기를 줄인다.
PNG는 무손실 압축을 사용하여 높은 화질을 유지하며 투명도를 지원한다.
GIF는 256색 제한이 있지만 애니메이션을 지원한다.
BMP는 압축하지 않은 이미지 데이터를 저장한다.

특히 이번 과제에서 사용되는 RAW는 다른 파일 형식과 달리 헤더 정보가 없으며,
8비트 단위로 밝기 값이 가로, 세로 크기를 곱한 화소 개수만큼 연속적으로 저장된다.

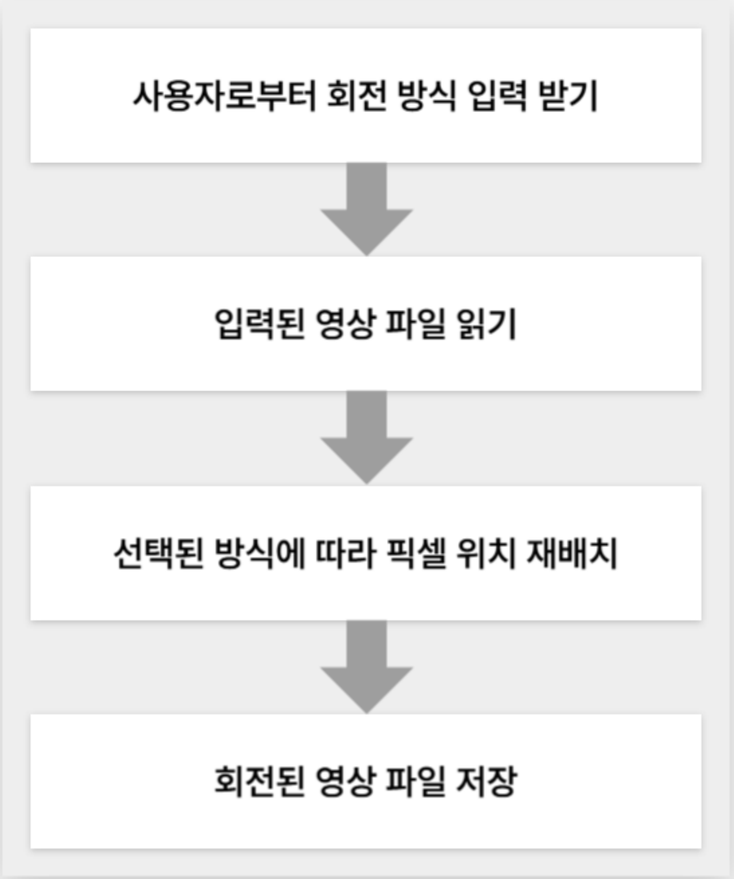
따라서 RAW 이미지는 일반적인 뷰어 프로그램으로는 읽을 수 없어서
RAW 이미지를 읽고 표시하기 위해서는 별도의 프로그램이 필요하게 된다.

[참고자료] "http://www.ktword.co.kr/test/view/view.php?m_temp1=3147"
"https://terms.naver.com/entry.naver?docId=5682324&cid=43667&categoryId=43667"
"https://www.scienceall.com/brd/board/390/L/menu/317?brdType=R&bbsSn=230521"

영상 회전 프로그램 구현

1. 프로그램 설계

이 프로그램은 사용자가 선택한 방식에 따라 영상을 회전 또는 반전시키는 기능을 가진다. 알고리즘은 아래와 같다.



2. 코드 구현 및 설명: 프로그램의 기본 설정

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define WIDTH 512
#define HEIGHT 512
#define INPUT_FILE "../../data/raw_image/l1.img"
#define OUTPUT_FILE "../../data/raw_image/l1"

int select(void);
unsigned char* load_image(const char* filename);
int rotate(int choice);
int save_image(const char* filename, unsigned char* img);

int main(void) {
    select();
}
```

1. '#include' 지시문
- '<stdio.h>'는 표준 입출력 함수를 사용하기 위한 헤더 파일이다.
- '<stdlib.h>'는 동적 메모리 할당 함수를 사용하기 위한 헤더 파일이다.

2. '#define' 매크로 정의
- '_CRT_SECURE_NO_WARNINGS'는 Visual Studio에서 보안 관련 경고를 비활성화하기 위해 사용한다. 주로 'scanf()'와 같은 함수 사용시 발생하는 경고를 방지한다.
- 'WIDTH', 'HEIGHT'는 영상의 너비와 높이 픽셀 값을 상수 값으로 정의한다.
- 'INPUT_FILE', 'OUTPUT_FILE'는 입출력 영상 파일의 기본 상대경로를 지정한다.
  이는 추후 경로 변경의 편의성을 위해 추가했다.

3. 함수 프로토타입
- 'select', 'load_image', 'rotate', 'save_image' 함수들의 선언부를 정의한다.

4. 'main()' 함수
- 'select' 함수를 호출한다.

2. 코드 구현 및 설명: 'select()' 함수

```
19  int select(void) {
20      int choice;
21      printf("1. Clockwise 90 | 2. Clockwise 180 | 3. Clockwise 270 | 4. Mirror | 5. Flip ₩n회전 방향을 선택하세요: ");
22      scanf("%d", &choice);
23
24      if (choice >= 1 && choice <= 5) {
25          return rotate(choice);
26      }
27      else {
28          printf("잘못된 선택입니다.");
29          return 1;
30      }
31  }
```

1. 변수 선언
- 사용자의 선택을 저장할 정수형 변수 'choice' 선언

2. 사용자 인터페이스
- 사용자에게 5가지의 선택 가능한 영상 변환 옵션을 표시

3. 사용자 입력 받기
- ‘scanf’를 사용해 사용자의 선택을 ‘choice’ 변수에 저장. ‘%d’ 형식 지정자는 정수 입력을 받기 위해 사용.

4. 입력 유효성 검사 및 처리
- 조건문으로 사용자 입력의 유효성을 검사. 유효한 입력인 경우, ‘rotate’ 함수를 호출해 선택된 작업을 수행.
  유효하지 않은 입력인 경우, 메시지를 출력한 뒤 1을 반환하여 함수를 종료.

2. 코드 구현 및 설명: 'load_image()' 함수

```
33  unsigned char* load_image(const char* filename) {
34      FILE* file_input = NULL;
35      unsigned char* img = NULL;
36
37      file_input = fopen(filename, "rb");
38      if (file_input == NULL) {
39          printf("파일이 존재하지 않습니다.₩n");
40          return NULL;
41      }
42
43      img = (unsigned char*)malloc(WIDTH * HEIGHT * sizeof(unsigned char));
44      if (img == NULL) {
45          printf("메모리 할당 오류 발생₩n");
46          fclose(file_input);
47          return NULL;
48      }
49
50      fread(img, sizeof(unsigned char), WIDTH * HEIGHT, file_input);
51      fclose(file_input);
52
53      return img;
54  }
```

1. 파일 열기

- 입력 파일을 바이너리 모드로 열고, 파일이 없으면 오류 메시지 출력.

2. 메모리 할당

- 원본 영상 'img'를 저장할 메모리를 동적 할당.

  * 'unsigned char'를 사용해 각 픽셀 값을 0~255 범위의 GrayScale 값으로 표현한다.

- 조건문을 사용해 메모리 할당 여부를 확인하고,
  할당 실패 시 오류 메시지를 출력하고 함수 종료.

3. 파일 입력

- 'fread'를 사용해 파일의 내용을 'img' 버퍼로 읽어들인 후 입력 파일 닫음.

2. 코드 구현 및 설명: 'rotate()' 함수

```c
int rotate(int choice) {
    unsigned char* img = NULL;
    unsigned char* img_rotate = NULL;

    img = load_image(INPUT_FILE);
    if (img == NULL) {
        return -1;
    }

    img_rotate = (unsigned char*)malloc(WIDTH * HEIGHT * sizeof(unsigned char));
    if (img_rotate == NULL) {
        printf("메모리 할당 오류 발생₩n");
        free(img);
        return -1;
    }

    for (int i = 0; i < HEIGHT; i++) {
        for (int j = 0; j < WIDTH; j++) {
            int new_i, new_j;
            switch (choice) {
            case 1: // Clockwise 90
                new_i = j;
                new_j = HEIGHT - 1 - i;
                break;
            case 2: // Clockwise 180
                new_i = HEIGHT - 1 - i;
                new_j = WIDTH - 1 - j;
                break;
            case 3: // Clockwise 270
                new_i = WIDTH - 1 - j;
                new_j = i;
                break;
            case 4: // Mirror
                new_i = i;
                new_j = WIDTH - 1 - j;
                break;
            case 5: // Flip
                new_i = HEIGHT - 1 - i;
                new_j = j;
                break;
            }
            *(img_rotate + new_i * WIDTH + new_j) = *(img + i * WIDTH + j);
        }
    }
```

1. 영상 데이터 할당 및 로드

- 각각 원본 영상과 회전된 영상의 픽셀 데이터를 저장할 img와 img_rotate 포인터 선언.
- 'load_image' 함수를 통해 원본 영상을 로드하고, 성공적으로 로드되지 않으면 -1을 반환.
- img_rotate에 대해 메모리를 할당하고,
  할당에 실패하면 오류 메시지를 출력하고 img의 메모리를 해제한 후 -1을 반환.

2. 영상 회전 및 변환

- 이중 반복문을 사용하여 원본 영상의 각 픽셀을 순회.
- switch문을 통해 사용자가 선택한 회전 방향에 따라 새로운 좌표 계산.

  * 각각을 함수화 시켜 사용해도 되지만, 코드의 간결화를 위해 하나의 switch문으로 작성함.

  ▶ Clockwise 90: 이는 x축을 y축으로, y축을 반대 방향으로 이동.

  ▶ Clockwise 180: 이는 x축과 y축 모두 반대 방향으로 이동.

  ▶ Clockwise 270: 이는 y축을 x축으로, x축을 반대 방향으로 이동.

  ▶ Mirror(수평 반전): 이는 x축을 고정하고 y축을 반대 방향으로 이동.

  ▶ Flip (수직 반전): 이는 y축을 고정하고 x축을 반대 방향으로 이동.

2. 코드 구현 및 설명: 'rotate()' 함수

```
101     char output_filename[256];
102     switch (choice) {
103     case 1: sprintf(output_filename, "%s_90.img", OUTPUT_FILE); break;
104     case 2: sprintf(output_filename, "%s_180.img", OUTPUT_FILE); break;
105     case 3: sprintf(output_filename, "%s_270.img", OUTPUT_FILE); break;
106     case 4: sprintf(output_filename, "%s_mirror.img", OUTPUT_FILE); break;
107     case 5: sprintf(output_filename, "%s_flip.img", OUTPUT_FILE); break;
108     }
109
110     if (save_image(output_filename, img_rotate) != 0) {
111         printf("이미지 저장 실패₩n");
112     }
113
114     free(img_rotate);
115     free(img);
116
117     return 0;
118 }
```

1. 출력 파일명 결정

- 선택된 변환 방식에 따라 'switch' 문을 통해 출력 파일명 설정.

2. 파일 출력

- 'save_image' 함수를 호출하여 변환된 영상을 파일로 저장.

3. 메모리 해제 및 종료

- 동적으로 할당된 메모리를 해제한 뒤 함수 종료.

코드 구현 및 설명: 'save_image()' 함수

```
120  int save_image(const char* filename, unsigned char* img) {
121      FILE* file_output = NULL;
122
123      file_output = fopen(filename, "wb");
124      if (file_output == NULL) {
125          printf("파일을 열 수 없습니다.₩n");
126          return -1;
127      }
128
129      fwrite(img, sizeof(unsigned char), WIDTH * HEIGHT, file_output);
130      fclose(file_output);
131
132      return 0;
133  }
```

1. 파일 열기

- 지정된 파일을 바이너리 쓰기 모드로 연다. 파일이 존재하지 않으면 새로 생성.
- 실패시 오류 메시지 출력하고 함수 종료.

2. 이미지 데이터 쓰기

- fwrite 함수로 이미지 데이터가 저장된 메모리 주소 img에서 파일 스트림 file_output으로 데이터를 쓴다.

3. 파일 닫기

- 시스템 리소스를 해제하는 단계로, 파일 스트림을 닫는다.

3. 프로그램 실행 결과 및 이미지 확인

본 과제에서 구현한 사진 회전 프로그램의 결과물인 Raw 이미지 파일은 교수님께서 제공해 주신 "RDisp.exe" 프로그램을 통해 확인할 수 있었다.
이 프로그램의 사용법은 별도로 안내되어 있으므로, 본 보고서에서는 회전된 이미지의 결과만을 제시한다.


원본


90도


180도


270도


Mirror


Flip

위의 이미지들은 각각 원본, 90도, 180도, 270도 회전, 좌우 반전(Mirror), 상하 반전(Flip)된 결과를 보여준다.
이를 통해 우리가 구현한 프로그램이 정상적으로 작동하여 원하는 이미지 변환을 수행했음을 확인할 수 있다.

Discussion

Discussion

영상 회전 프로그램 제작 과제를 통해 C언어의 핵심 개념을 학습하고 영상 처리의 기본 원리와 RAW 영상 파일의 특성에 대한 이해를 넓힐 수 있었다.

그러나 현재 구현된 기능을 넘어 프로그램의 확장 가능성에 대해 다양한 측면에서 검토할 필요가 있었다.

특히 다양한 파일 형식 지원과 더 정밀한 회전 기능 구현은 프로그램의 실용성을 크게 향상시킬 수 있는 중요한 요소이기 때문이다.

따라 다음과 같은 사항들에 대해 Discussion해보고 싶다.

1. 다양한 이미지 파일 형식 지원
2. 정밀한 이미지 회전 기능 구현

Discussion, 1. 다양한 이미지 파일 형식 지원

현재 헤더가 없는 RAW 파일만을 처리할 수 있으나, 실제 환경에서는 다양한 이미지 파일 형식이 사용되고 있기에 이에 대한 지원이 필요하다.
파일 형식마다 고유한 헤더 구조와 데이터 저장 방식을 가지고 있어, 이를 효과적으로 처리하기 위한 아래와 같은 방안을 고안해볼 수 있었다.

1) 파일 헤더 처리 코드 구현

헤더에는 이미지의 너비, 높이, 색상 깊이, 압축 방식 등 중요한 메타데이터가 저장되어 있다.
따라서 각 파일 형식별 헤더 구조를 이해하고 적절히 처리하는 코드를 직접 구현함으로써 다양한 파일 형식을 지원할 수 있을 것이다.

예를 들어, BMP 파일의 경우 파일 헤더(14바이트)와 정보 헤더(40바이트)로 구성되어 있는데, 이를 구조체로 정의하여 파일에서 읽어들인 후 처리할 수 있다. 이러한 접근 방식은 PNG, GIF 등 다른 파일 형식에도 확장 적용할 수 있을 것다.

2) 이미지 처리 라이브러리 활용

C언어에서 다양한 파일 형식을 사용하기 위해 여러 라이브러리를 활용할 수 있었다.
대표적으로 JPEG 이미지 처리를 위한 libjpeg, PNG 파일을 위한 libpng, 다양한 RAW 포맷을 지원하는 LibRaw 등이 있다.
이러한 라이브러리들은 쉽게 이미지 데이터에 접근할 수 있도록 API를 제공하게 된다.

[참고자료] "https://ko.wikipedia.org/wiki/BMP_%ED%8C%8C%EC%9D%BC_%ED%8F%AC%EB%A7%B7"
"https://cafe.naver.com/cortexsh/5251?art=ZXh0ZXJuYWwtc2VydmljZS1uYXZlci1zZWFyY2gtY2FmZS1wcg.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjYWZlVHlwZSI6IkNBRkVfVVJMIiwiY2FmZVVybCI6ImNvcnRleHNoIiwiYXJ0aWNsZUlkIjo1MjUxLCJpc3N1ZWRBdCI6MTc0MTgwMTUyNTU3MX0.TinJh3ONfBXp7VrU6myTsqisg5SFgKtneDQID5eRoFk"
"https://ko.wikipedia.org/wiki/Libjpeg"

Discussion, 2. 정밀한 이미지 회전 기능 구현

현재 프로그램은 90도 단위 회전만 지원하고 있으나, 임의 각도의 회전이 필요한 경우도 있을 것이다.
90도 단위 회전은 단순한 픽셀 위치 재배열로 가능하지만, 임의 각도 회전은 더 복잡한 수학적 계산과 이미지 처리 기법이 필요했다.

1) 회전 행렬 기반 변환 알고리즘

임의 각도 회전은 2D 회전 행렬 [cos(θ) -sin(θ); sin(θ) cos(θ)]을 이용한 좌표 변환을 통해 구현할 수 있다.
이미지 중심(cx, cy)을 기준으로 회전할 경우, 픽셀 (x, y)의 새 위치는 오른쪽과 같이 계산된다:
이 변환을 모든 픽셀에 적용하여 회전된 이미지를 생성한다.

$$x' = (x-cx)\cos\theta - (y-cy)\sin\theta + cx$$

$$y' = (x-cx)\sin\theta + (y-cy)\cos\theta + cy$$

2) 픽셀 보간법 구현

임의 각도 회전 시 계산된 좌표는 정수가 아닌 경우가 많아, 가장 가까운 정수 좌표의 픽셀 값을 사용하거나,
더 나은 화질을 위해서는 주변 4개 픽셀의 가중 평균을 계산하여 더 부드러운 결과를 제공하는 쌍선형 보간법을 사용한다.

3) 경계 처리 및 이미지 크기 조정

임의 각도 회전 시 원본 이미지의 경계를 벗어나는 부분이 발생하거나, 회전 후 이미지 크기가 달라질 수 있다. 이러한 문제를 해결하기 위해:
첫째, 원본 이미지의 크기를 유지하면서 경계를 벗어나는 부분은 검은색이나 특정 색상으로 채우는 방법
둘째, 회전 후 모든 픽셀이 포함될 수 있도록 출력 이미지 크기를 자동으로 조정하는 방법 등이 있다.

[참고자료] "https://blog.naver.com/liveforu/222030359973"
"https://blog.naver.com/khlee0821/222500854466"

YouTube 과제 영상 링크

https://youtu.be/jXuEdkUIlD0?si=CY5nIcrsD66ww5YA