**TEAM PROJECT**

**[Computer Programming Ⅱ 1분반]**

Team name: **Pumpkin**

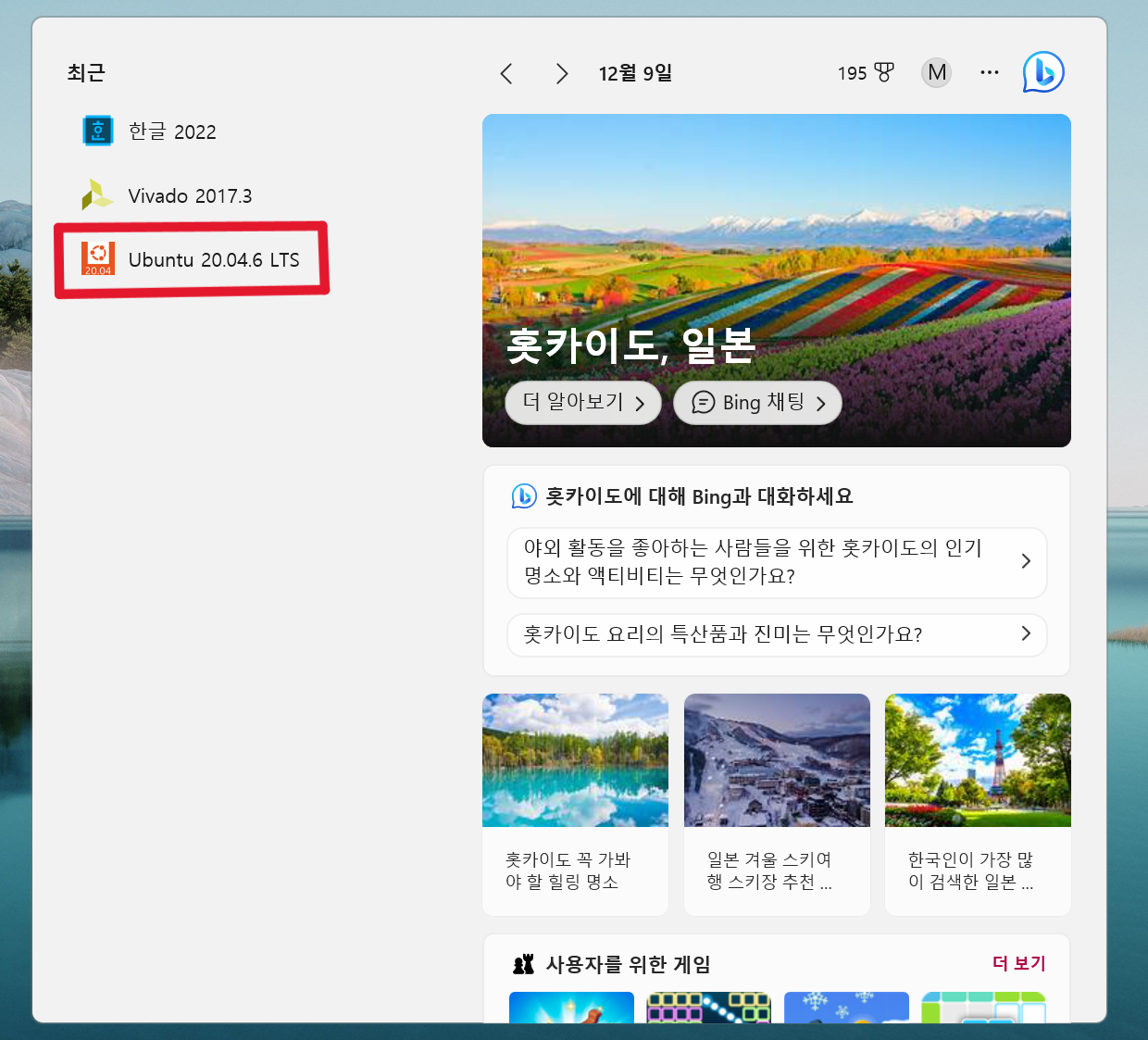
Team member: **20212020 박민준, 20212021 원대호**

**TEAM PROJET APPLICATION AND IMPLEMENTATIONS**

**-LET’S FLIP THE IMAGE UPSIDE DOWN-**

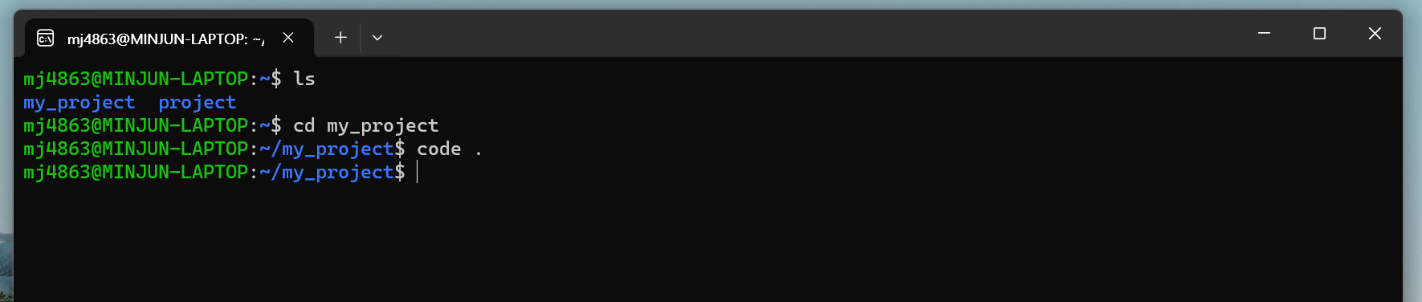
**[1] How to setup our development environment**

**[A] Ubuntu 실행**

****

**개발 환경으로 Ubuntu 20.04.6 LTS를 이용하였다.**

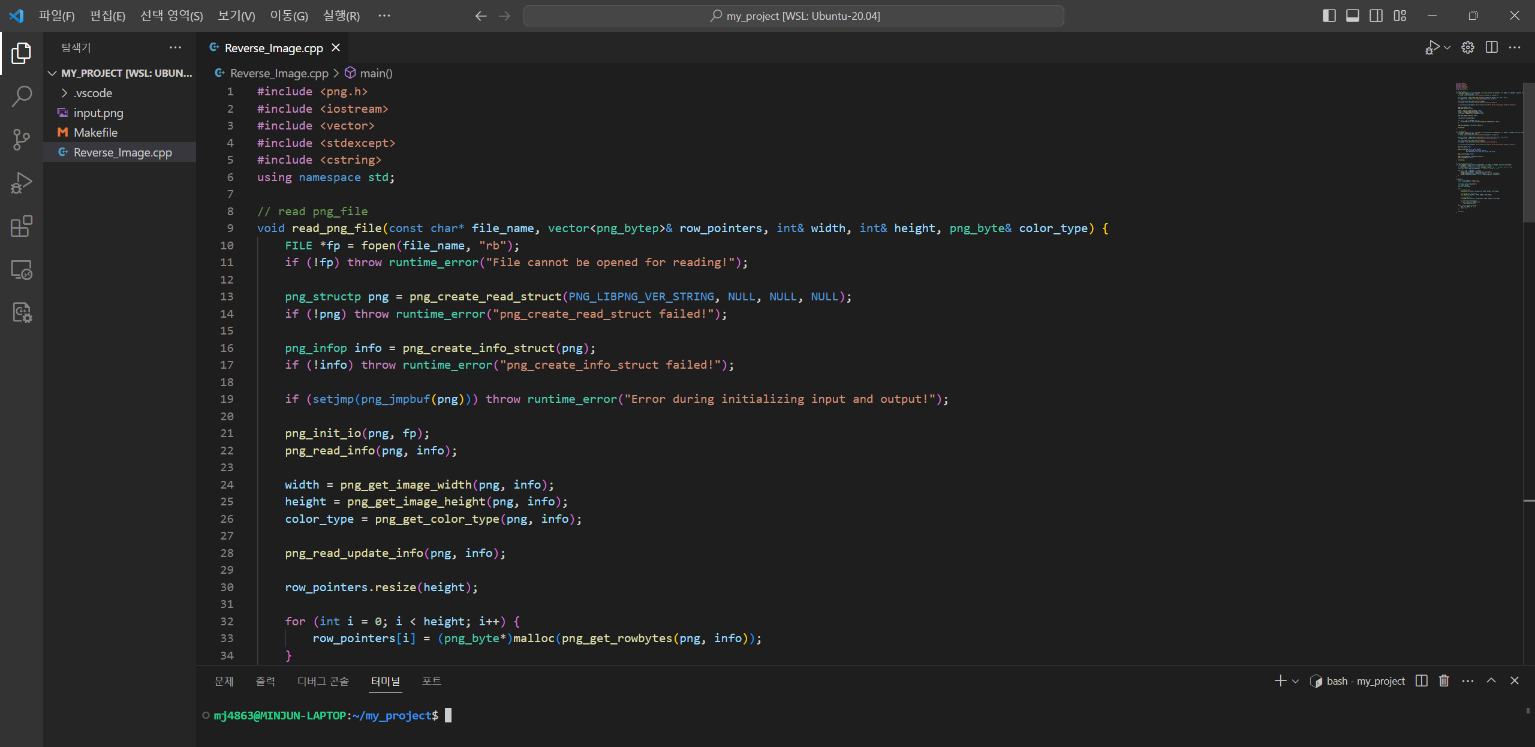
**[B] VS Code 실행**

****

**‘mkdir my\_project’ 명령어를 이용해 생성한 my\_project directory에서 작업을 진행하였다.**

**‘cd my\_project’ 명령어를 이용해 directory를 이동하였고, ‘code .’ 명령어를 통해 VS Code를 실행하였다.**

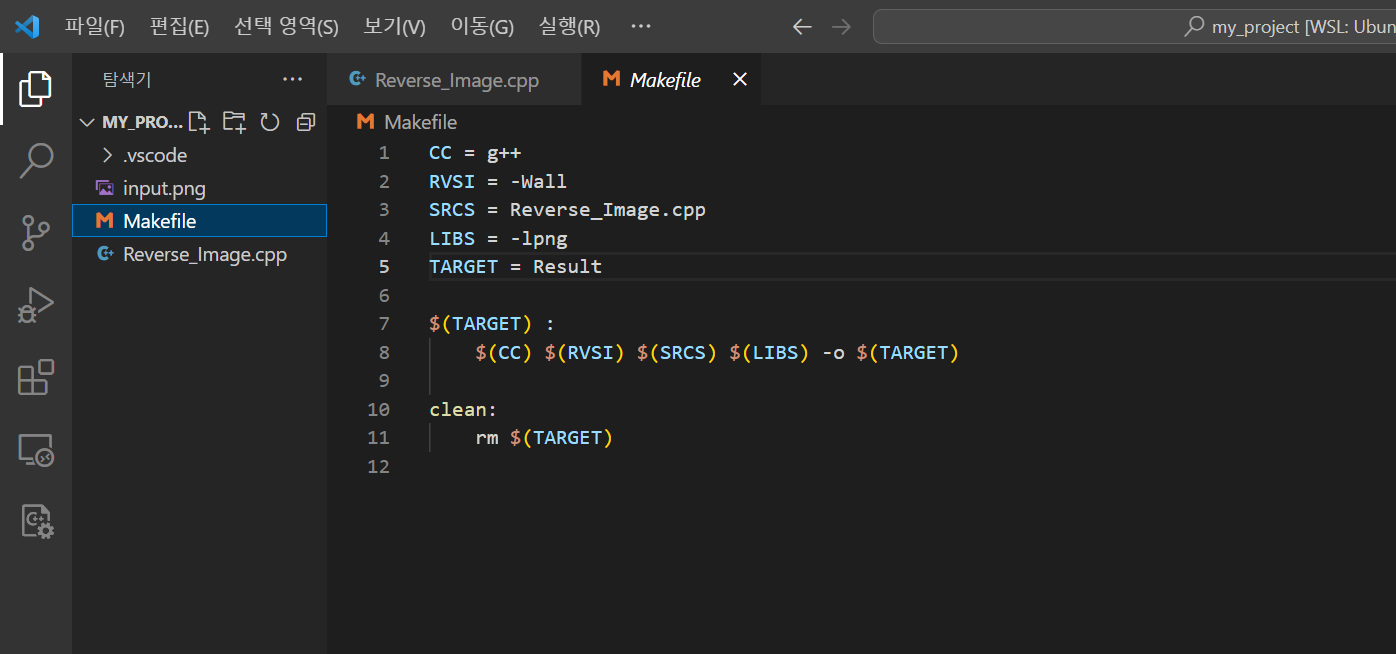
**[C] 이미지를 수직으로 뒤집는 Code 작성**

****

**input.png 이미지를 읽어들이고, 해당 이미지를 수직으로 뒤집어서 다시 output.png 파일로 출력하는 코드를 작성하였다.**

**해당 코드는 뒤에서 자세히 설명하고자 한다.**

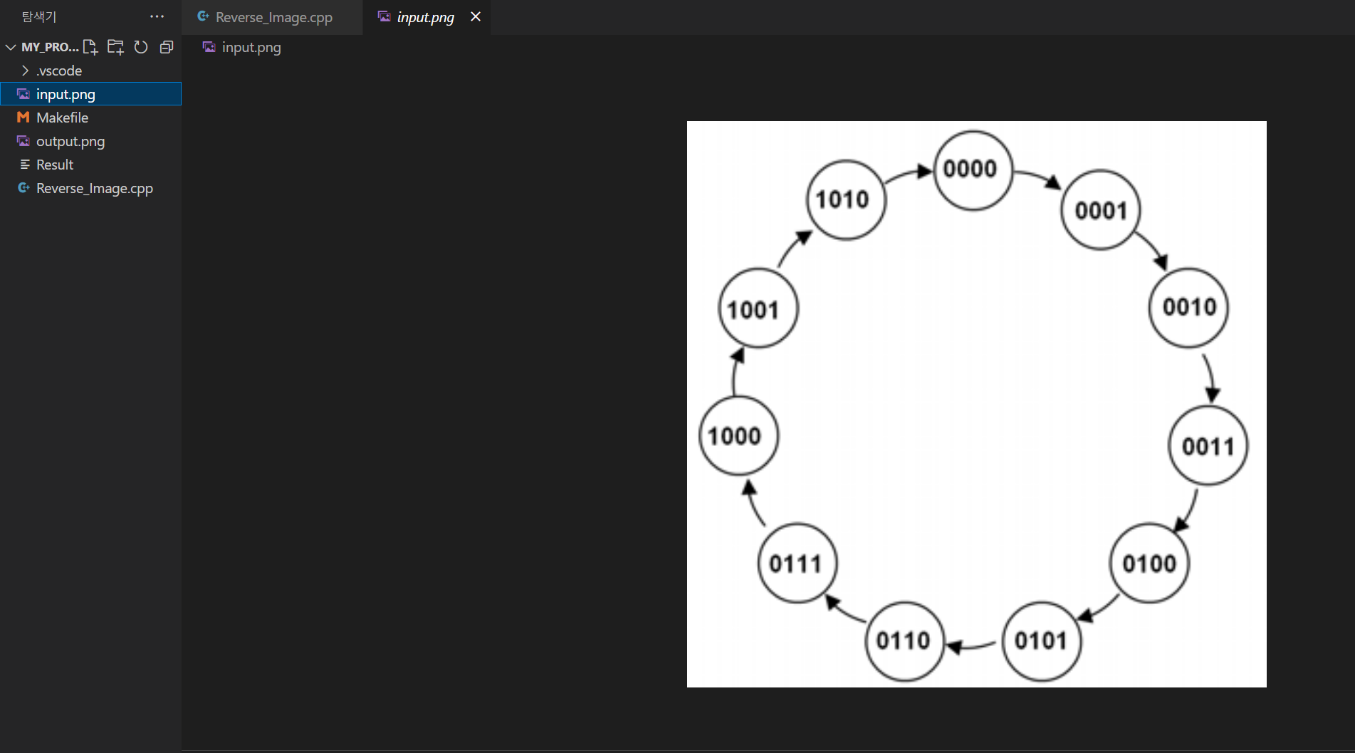
**[D] Makefile 생성**

****

**make 명령어를 수행할 Makefile을 생성하였다.**

**해당 Makefile은 Reverse\_Image.cpp 파일을 이용해 Result 실행 파일을 만드는 것이 목적이다.**

**[E] input.png 파일 선정**

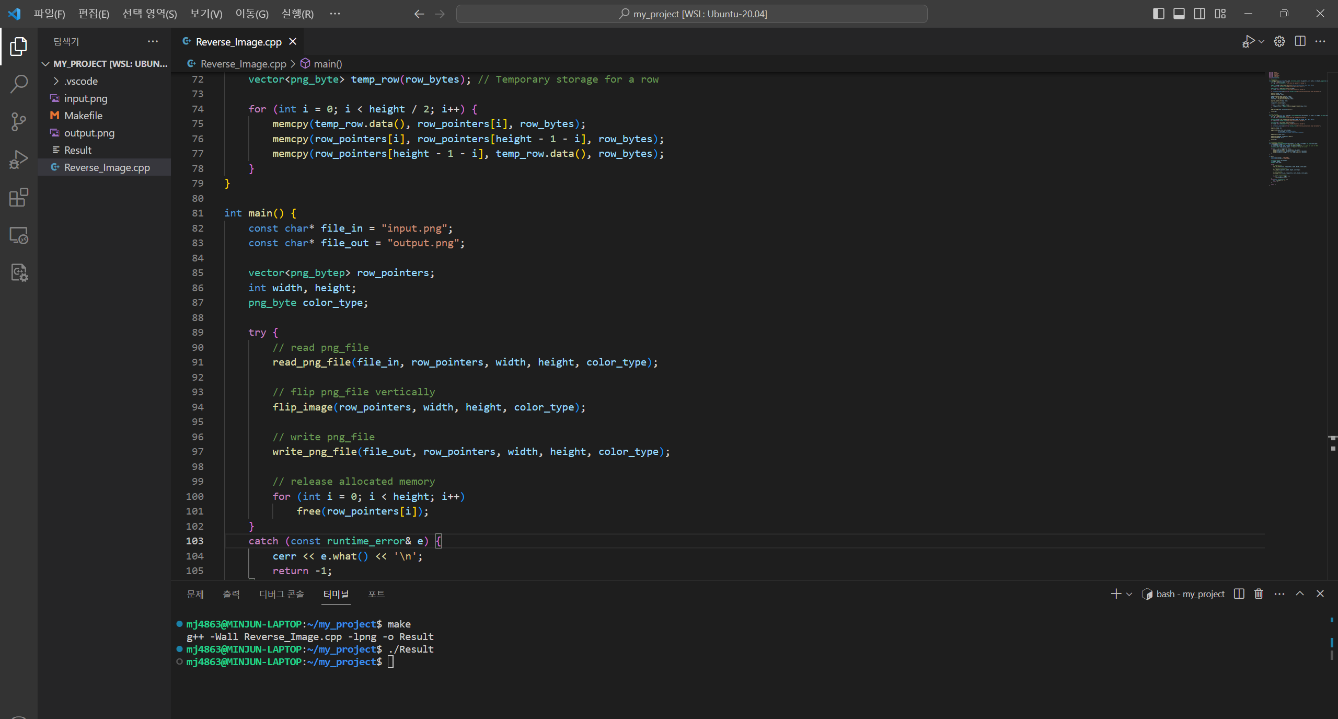
****

**수직으로 반전시킬 input.png 파일을 선정하였다.**

**해당 이미지는 4bit decade counter에 해당하는 이미지이다.**

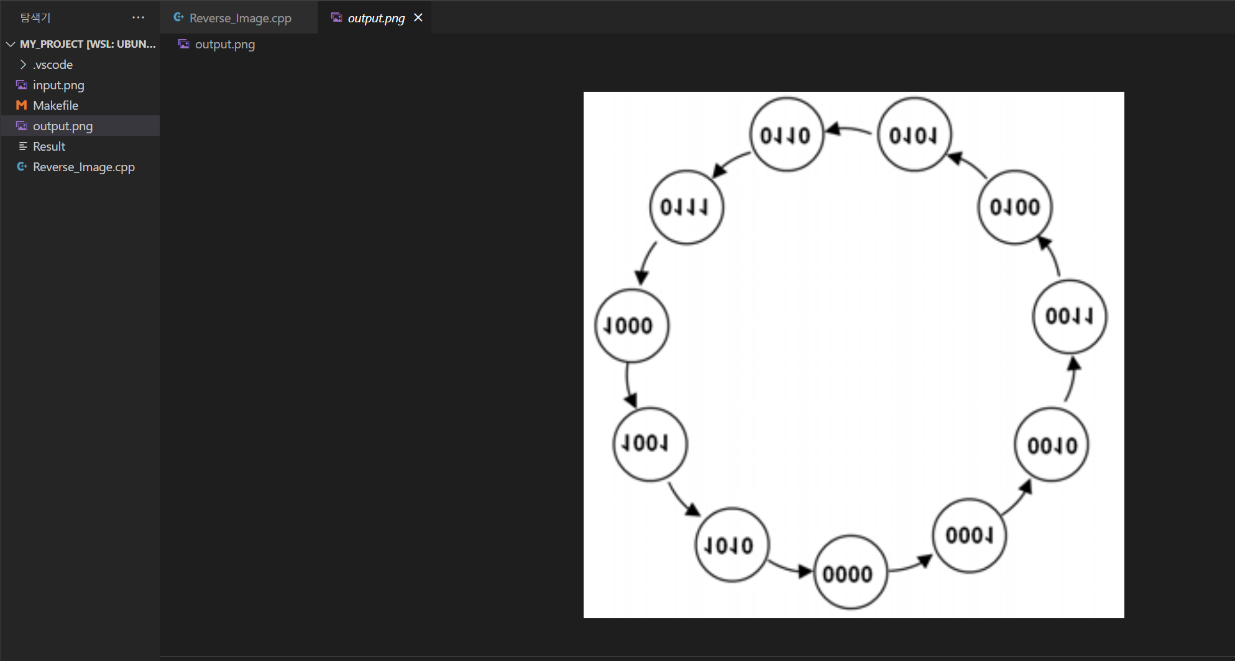
**[2] How to implement our program**

**[A] make 명령어 실행 후 생성된 실행 파일 재실행**

****

**터미널 창에서 make 명령어를 통해 Makefile을 실행하였다. 그 결과 왼쪽 탐색기 창을 확인하면Result 실행 파일이 생성된 것을 알 수 있다. 이후 ‘./Result’ 명령어를 통해 해당 실행 파일을 재실행함으로써 output.png 파일을 출력할 수 있다.**

**[B] 이미지 반전 결과**

****

**이미지 반전 결과는 다음과 같다. input.png 파일이 위아래로 뒤집어진 것을 확인할 수 있다.**

**[3] Full Code**

**#include <png.h>**

**#include <iostream>**

**#include <vector>**

**#include <stdexcept>**

**#include <cstring>**

**using namespace std;**

**// read png\_file**

**void read\_png\_file(const char\* file\_name, vector<png\_bytep>& row\_pointers, int& width, int& height, png\_byte& color\_type) {**

**FILE \*fp = fopen(file\_name, "rb");**

**if (!fp) throw runtime\_error("File cannot be opened for reading!");**

**png\_structp png = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);**

**if (!png) throw runtime\_error("png\_create\_read\_struct failed!");**

**png\_infop info = png\_create\_info\_struct(png);**

**if (!info) throw runtime\_error("png\_create\_info\_struct failed!");**

**if (setjmp(png\_jmpbuf(png))) throw runtime\_error("Error during initializing input and output!");**

**png\_init\_io(png, fp);**

**png\_read\_info(png, info);**

**width = png\_get\_image\_width(png, info);**

**height = png\_get\_image\_height(png, info);**

**color\_type = png\_get\_color\_type(png, info);**

**png\_read\_update\_info(png, info);**

**row\_pointers.resize(height);**

**for (int i = 0; i < height; i++) {**

**row\_pointers[i] = (png\_byte\*)malloc(png\_get\_rowbytes(png, info));**

**}**

**png\_read\_image(png, row\_pointers.data());**

**fclose(fp);**

**}**

**// write png\_file**

**void write\_png\_file(const char\* file\_name, vector<png\_bytep>& row\_pointers, int width, int height, png\_byte color\_type) {**

**FILE \*fp = fopen(file\_name, "wb");**

**if (!fp) throw runtime\_error("File cannot be opened for writing!");**

**png\_structp png = png\_create\_write\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);**

**if (!png) throw runtime\_error("png\_create\_write\_struct failed!");**

**png\_infop info = png\_create\_info\_struct(png);**

**if (!info) throw runtime\_error("png\_create\_info\_struct failed!");**

**if (setjmp(png\_jmpbuf(png))) throw runtime\_error("Error during initializing input and output");**

**png\_init\_io(png, fp);**

**png\_set\_IHDR(png, info, width, height,**

**8, color\_type, PNG\_INTERLACE\_NONE,**

**PNG\_COMPRESSION\_TYPE\_BASE, PNG\_FILTER\_TYPE\_BASE);**

**png\_write\_info(png, info);**

**png\_write\_image(png, row\_pointers.data());**

**png\_write\_end(png, NULL);**

**fclose(fp);**

**}**

**//flip png\_file vertically**

**void flip\_image(vector<png\_bytep>& row\_pointers, int width, int height, png\_byte color\_type) {**

**int row\_bytes = width \* 4; // 4 bytes per pixel for RGBA**

**if (color\_type == PNG\_COLOR\_TYPE\_RGB) row\_bytes = width \* 3; // 3 bytes per pixel for RGB**

**vector<png\_byte> temp\_row(row\_bytes); // Temporary storage for a row**

**for (int i = 0; i < height / 2; i++) {**

**memcpy(temp\_row.data(), row\_pointers[i], row\_bytes);**

**memcpy(row\_pointers[i], row\_pointers[height - 1 - i], row\_bytes);**

**memcpy(row\_pointers[height - 1 - i], temp\_row.data(), row\_bytes);**

**}**

**}**

**int main() {**

**const char\* file\_in = "input.png";**

**const char\* file\_out = "output.png";**

**vector<png\_bytep> row\_pointers;**

**int width, height;**

**png\_byte color\_type;**

**try {**

**// read png\_file**

**read\_png\_file(file\_in, row\_pointers, width, height** **, color\_type);**

**// flip png\_file vertically**

**flip\_image(row\_pointers, height,** **color\_type);**

**// write png\_file**

**write\_png\_file(file\_out, row\_pointers, width, height, color\_type);**

**// release allocated memory**

**for (int i = 0; i < height; i++)**

**free(row\_pointers[i]);**

**}**

**catch (const runtime\_error& e) {**

**cerr << e.what() << '\n';**

**return -1;**

**}**

**return 0;**

**}**

**[4] CODE DESCRIPTION**

**[A] 헤더 파일 선언**

**#include <png.h>**

**#include <iostream>**

**#include <vector>**

**#include <stdexcept>**

**#include <cstring>**

**using namespace std;**

**이 코드는 C++ 프로그램을 동작할 때 필요한 헤더 파일들을 include하고, 표준 output을 위한 iostream과 동적 배열을 다루기 위한 vector, 예외 처리를 위한 stdexcept를 이용하겠다는 것을 나타낸다. 각각의 부분에 대해 좀 더 자세히 설명하겠다.**

**[1] #include <png.h>**

**PNG 이미지를 다루기 위한 헤더 파일이다. png.h 헤더는 PNG 이미지를 읽고 쓰기 위한 함수와 데이터 구조체 등을 정의하고 있다.**

**[2] #include <iostream>**

**C++의 표준 입력 및 출력을 위한 헤더 파일이다. iostream 헤더를 사용하면 cout, cin, cerr 등을 사용하여 표준 출력과 입력을 다룰 수 있다.**

**[3] #include <vector>**

**동적 배열인 vector를 사용하기 위한 헤더 파일이다. vector는 배열과 유사하지만 크기를 동적으로 조절할 수 있으며, 여러 유용한 기능을 제공한다.**

**[4] #include <stdexcept>**

**예외 처리를 위한 표준 예외 클래스들을 정의하는 헤더 파일이다. 여기서는 runtime\_error를 사용하여 예외를 처리한다.**

**[5]#include<cstring>**

**<cstring> 헤더 파일은 다양한 문자열 관련 함수와 메모리 조작 함수를 제공한다. 여기에는 strcpy, strcat, strlen, memcpy, memset 등이 포함되어 있다. 이러한 함수들은 문자열 복사, 이어 붙이기, 길이 계산, 메모리 복사, 메모리 초기화 등 다양한 작업을 수행하는 데 사용된다.**

**[6] using namespace std;**

**std namespace를 사용한다는 것을 선언한다. C++에서는 표준 라이브러리 함수와 객체들이 std라는 네임스페이스에 정의되어 있다. using namespace std;를 사용하면 cout, cin, vector 등을 std::를 붙이지 않고 사용할 수 있다.**

**[B] main 함수**

**int main() {**

**const char\* file\_in = "input.png";**

**const char\* file\_out = "output.png";**

**vector<png\_bytep> row\_pointers;**

**int width, height;**

**png\_byte color\_type;**

**try {**

**// read png\_file**

**read\_png\_file(file\_in, row\_pointers, width, height, color\_type);**

**// flip png\_file vertically**

**flip\_image(row\_pointers, height, color\_type);**

**// write png\_file**

**write\_png\_file(file\_out, row\_pointers, width, height, color\_type);**

**// release allocated memory**

**for (int i = 0; i < height; i++)**

**free(row\_pointers[i]);**

**}**

**catch (const runtime\_error& e) {**

**cerr << e.what() << '\n';**

**return -1;**

**}**

**return 0;**

**}**

**[1] main 함수**

**int main() {**

**const char\* file\_in = "input.png";**

**const char\* file\_out = "output.png";**

**vector<png\_bytep> row\_pointers;**

**int width, height;**

**png\_byte color\_type;**

**이 코드는 PNG 이미지를 읽어들여 수직으로 뒤집은 후, 새로운 파일에 저장하는 C++ 프로그램의 main 함수이다. main code를 한 줄씩 설명하고자 한다. main 함수는 입력 파일(input.png)과 출력 파일(output.png)의 경로를 정의하고 있다. 또한 이미지의 행을 저장할 벡터(row\_pointers) 및 이미지의 너비(width)와 높이(height)를 선언하였다.**

**vector<png\_bytep>는 C++ 표준 라이브러리에서 제공하는 vector 컨테이너를 사용하고 있다. 이 벡터는 png\_bytep라는 타입의 포인터를 저장한다. 이때 png\_bytep는 PNG 이미지의 한 행을 나타내는 자료형이다. PNG 이미지의 각 행은 byte의 배열로 표현되며, png\_bytep는 이러한 바이트 배열을 가리키는 포인터이다.**

**이에 vector<png\_bytep>는 PNG 이미지의 각 행에 대한 포인터를 동적으로 저장하는데 사용된다. 코드에서 이 벡터는 이미지를 읽거나 쓰는 과정에서 각 행의 데이터를 저장하는 데 활용된다.**

**png\_byte color\_type;는 PNG 이미지의 color type을 표현하는 변수이다.**

**[2] 이미지 읽기**

**try {**

**// read png\_file**

**read\_png\_file(file\_in, row\_pointers, width, height, color\_type);**

**try 블록 내에서 read\_png\_file 함수를 호출하여 입력 PNG 이미지를 읽어들인다.**

**이때 file\_in은 입력 파일의 경로를 나타내고, row\_pointers, width, height, color\_type는 이미지의 행 데이터, 너비, 높이, 컬러 타입을 저장할 변수들이다.**

**[3] 이미지 뒤집기**

**// flip png\_file vertically**

**flip\_image(row\_pointers, height,** **color\_type);**

**flip\_image 함수를 호출하여 이미지를 수직으로 뒤집는다. row\_pointers 벡터와 이미지의 높이(height) 그리고 이미지의 컬러 타입을 전달한다.**

**[4] 이미지 쓰기**

**// write png\_file**

**write\_png\_file(file\_out, row\_pointers, width, height,** **color\_type);**

**write\_png\_file 함수를 호출하여 뒤집힌 이미지를 출력 파일(output.png)에 쓴다. 이때row\_pointers, width, height, color\_type를 전달한다.**

**[5] 메모리 할당 해제**

**// release allocated memory**

**for (int i = 0; i < height; i++)**

**free(row\_pointers[i]);**

**이미지를 읽을 때 동적으로 할당된 메모리를 해제한다. free 함수를 사용하여 각 행의 메모리를 해제한다. 이 부분은 이미지를 읽을 때 동적으로 할당된 메모리를 명시적으로 해제하는 부분이다. code에서 malloc 함수를 이용해 각 이미지 행에 대한 메모리를 동적으로 할당하고 있다.**

**EX) row\_pointers[i] = (png\_byte\*)malloc(png\_get\_rowbytes(png, info));**

**이렇게 동적으로 할당된 메모리는 프로그램이 종료되기 전에 반드시 해제되어야 한다. 그렇지 않으면 메모리 누수가 발생하게 돼 시스템 자원이 낭비된다.**

**[6] 예외 처리**

**catch (const runtime\_error& e) {**

**cerr << e.what() << '\n';**

**return -1;**

**}**

**try 블록에서 발생한 예외(runtime\_error)를 catch 블록에서 처리한다. 예외가 발생하면 에러 메시지를 출력하고 프로그램을 -1로 종료한다. 예외 처리는 프로그램이 실행 중에 예상치 못한 상황이나 오류가 발생했을 때 해당 상황을 처리하고 프로그램의 안정성을 유지하는 데 도움이 된다.**

**[7] 프로그램 종료**

**return 0;**

**[C] 파일 읽기**

**// read png\_file**

**void read\_png\_file(const char\* file\_name, vector<png\_bytep>& row\_pointers, int& width, int& height, png\_byte& color\_type) {**

**FILE \*fp = fopen(file\_name, "rb");**

**if (!fp) throw runtime\_error("File cannot be opened for reading!");**

**png\_structp png = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);**

**if (!png) throw runtime\_error("png\_create\_read\_struct failed!");**

**png\_infop info = png\_create\_info\_struct(png);**

**if (!info) throw runtime\_error("png\_create\_info\_struct failed!");**

**if (setjmp(png\_jmpbuf(png))) throw runtime\_error("Error during initializing input and output!");**

**png\_init\_io(png, fp);**

**png\_read\_info(png, info);**

**width = png\_get\_image\_width(png, info);**

**height = png\_get\_image\_height(png, info);**

**color\_type = png\_get\_color\_type(png, info);**

**png\_read\_update\_info(png, info);**

**row\_pointers.resize(height);**

**for (int i = 0; i < height; i++) {**

**row\_pointers[i] = (png\_byte\*)malloc(png\_get\_rowbytes(png, info));**

**}**

**png\_read\_image(png, row\_pointers.data());**

**fclose(fp);**

**}**

**[1] 함수 선언**

**// read png\_file**

**void read\_png\_file(const char\* file\_name, vector<png\_bytep>& row\_pointers, int& width, int& height, png\_byte& color\_type)) {**

**read\_png\_file 함수를 선언하며, PNG 파일을 읽어서 정보를 추출하고 이미지 데이터를 row\_pointers 벡터에 저장하는 역할을 한다.**

**매개변수로는 파일 이름(file\_name), 행 data를 저장할 벡터(row\_pointers), 이미지의 너비(width), 그리고 높이(height), 이미지의 컬러 타입을 받는다.**

**[2] 파일 열기 및 예외 처리**

**FILE \*fp = fopen(file\_name, "rb");**

**if (!fp) throw runtime\_error("File cannot be opened for reading!");**

**fopen 함수를 사용하여 파일을 연다. 파일이 성공적으로 열리지 않으면 파일을 여는 데 실패했음을 표현하는 예외를 던진다.**

**[3] 읽기 구조체 초기화 및 예외 처리**

**png\_structp png = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);**

**if (!png) throw runtime\_error("png\_create\_read\_struct failed!");**

**png\_create\_read\_struct 함수를 사용해 PNG 읽기 구조체를 초기화한다. 만약 초기화에 실패하면 해당 예외를 던진다.**

**[4] 예외 환경 처리 설정**

**if (setjmp(png\_jmpbuf(png))) throw runtime\_error("Error during initializing input and output!");**

**setjmp 함수를 사용해 예외 처리를 위한 환경을 set한다. 만약 초기화에 실패하면 해당 예외를 던진다.**

**[5] PNG 입출력 초기화 및 이미지 정보 읽기**

**png\_init\_io(png, fp);**

**png\_read\_info(png, info);**

**png\_init\_io 함수로 PNG 입출력을 초기화한다. 또한 png\_read\_info 함수로 이미지의 정보를 읽어온다.**

**[6] 이미지 너비와 높이, 컬러 타입 설정**

**width = png\_get\_image\_width(png, info);**

**height = png\_get\_image\_height(png, info);**

**color\_type = png\_get\_color\_type(png, info);**

**png\_get\_image\_width와 png\_get\_image\_height 함수로 이미지의 너비와 높이를 설정한다.**

**color\_type = png\_get\_color\_type(png, info);는 libpng library function을 이용해 PNG image의 컬러 타입을 얻어와서 color\_type variable에 저장하는 code이다.**

**[7] 이미지 정보 업데이트**

**png\_read\_update\_info(png, info);**

**png\_read\_update\_info 함수로 이미지의 정보를 업데이트한다.**

**[8] 이미지 행에 대한 동적 메모리 할당**

**row\_pointers.resize(height);**

**for (int i = 0; i < height; i++) {**

**row\_pointers[i] = (png\_byte\*)malloc(png\_get\_rowbytes(png, info));**

**}**

**row\_pointers 벡터의 크기를 이미지의 높이로 재조정한다. 또한 각 행에 대한 동적 메모리를 할당하여 row\_pointers에 저장한다.**

**[9] 이미지 데이터 읽기 및 파일 닫기**

**png\_read\_image(png, row\_pointers.data());**

**fclose(fp);**

**}**

**png\_read\_image 함수를 호출하여 image data를 읽어와 row\_pointers에 저장한다.**

**fclose 함수를 호출해 열었던 file을 닫는다.**

**[D] 파일 쓰기**

**// write png\_file**

**void write\_png\_file(const char\* file\_name, vector<png\_bytep>& row\_pointers, int width, int height, 3. png\_byte color\_type) {**

**FILE \*fp = fopen(file\_name, "wb");**

**if (!fp) throw runtime\_error("File cannot be opened for writing!");**

**png\_structp png = png\_create\_write\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);**

**if (!png) throw runtime\_error("png\_create\_write\_struct failed!");**

**png\_infop info = png\_create\_info\_struct(png);**

**if (!info) throw runtime\_error("png\_create\_info\_struct failed!");**

**if (setjmp(png\_jmpbuf(png))) throw runtime\_error("Error during initializing input and output");**

**png\_init\_io(png, fp);**

**png\_set\_IHDR(png, info, width, height,**

**8, 4. color\_type, PNG\_INTERLACE\_NONE,**

**PNG\_COMPRESSION\_TYPE\_BASE, PNG\_FILTER\_TYPE\_BASE);**

**png\_write\_info(png, info);**

**png\_write\_image(png, row\_pointers.data());**

**png\_write\_end(png, NULL);**

**fclose(fp);**

**}**

**[1] 함수 선언**

**// write png\_file**

**void write\_png\_file(const char\* file\_name, vector<png\_bytep>& row\_pointers, int width, int height, png\_byte color\_type) {**

**write\_png\_file 함수를 선언하며, PNG 파일에 image data를 쓰는 역할을 한다.**

**매개변수로는 파일 이름(file\_name), 행 데이터를 저장한 벡터(row\_pointers), 이미지의 너비(width), 그리고 높이(height), 이미지의 컬러 타입을 받는다.**

**[2] 파일 열기 및 예외 처리**

**FILE \*fp = fopen(file\_name, "wb");**

**if (!fp) throw runtime\_error("File cannot be opened for writing!");**

**fopen 함수를 사용해 파일을 쓰기 모드로 연다. file이 성공적으로 열리지 않으면 파일을 열지 못했다는 예외를 던진다.**

**[3] 쓰기 구조체 초기화 및 예외 처리**

**png\_structp png = png\_create\_write\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);**

**if (!png) throw runtime\_error("png\_create\_write\_struct failed!");**

**png\_create\_write\_struct 함수를 사용해 PNG 쓰기 구조체를 초기화한다. 초기화에 실패하면 해당 예외를 던진다.**

**[4] 정보 구조체 초기화 및 예외 처리**

**png\_infop info = png\_create\_info\_struct(png);**

**if (!info) throw runtime\_error("png\_create\_info\_struct failed!");**

**png\_create\_info\_struct 함수를 사용해 PNG 정보 구조체를 초기화한다. 초기화에 실패하면 해당 예외를 던진다.**

**[5] 예외 처리 환경 설정**

**if (setjmp(png\_jmpbuf(png))) throw runtime\_error("Error during initializing input and output");**

**setjmp 함수를 사용하여 예외 처리를 위한 환경을 set한다. 초기화에 실패하면 해당 예외를 던진다.**

**[6] PNG 입출력 초기화**

**png\_init\_io(png, fp);**

**png\_init\_io 함수로 PNG 입출력을 초기화한다. file pointer(fp)를 사용해 초기화한다.**

**[7]이미지 헤더 정보 설정**

**png\_set\_IHDR(png, info, width, height,**

**8, PNG\_COLOR\_TYPE\_RGBA, PNG\_INTERLACE\_NONE,**

**PNG\_COMPRESSION\_TYPE\_BASE, PNG\_FILTER\_TYPE\_BASE);**

**png\_set\_IHDR 함수로 image의 헤더 정보를 set한다. 여기서는 너비(width), 높이(height), 비트 수(8), 컬러 타입(PNG\_COLOR\_TYPE\_RGBA), 인터레이스 모드(PNG\_INTERLACE\_NONE), 압축 타입(PNG\_COMPRESSION\_TYPE\_BASE), 필터 타입(PNG\_FILTER\_TYPE\_BASE)을 set한다.**

**[8]PNG 정보 쓰기**

**png\_write\_info(png, info);**

**png\_write\_info 함수를 호출해 image의 정보를 쓴다.**

**[9] 이미지 데이터 쓰기**

**png\_write\_image(png, row\_pointers.data());**

**png\_write\_image 함수로 이미지 데이터를 쓴다. row\_pointers.data()를 통해 벡터에 저장된 image data의 포인터를 전달한다.**

**[10] PNG 쓰기 완료**

**png\_write\_end(png, NULL);**

**png\_write\_end 함수를 호출해 PNG 쓰기 작업을 끝낸다.**

**[11]**

**파일 닫기**

**fclose(fp);**

**}**

**fclose 함수로 열려있던 파일을 닫는다.**

**[E] 이미지 뒤집기**

**//flip png\_file vertically**

**void flip\_image(vector<png\_bytep>& row\_pointers, int width, int height, png\_byte color\_type) {**

**int row\_bytes = width \* 4; // 4 bytes per pixel for RGBA**

**if (color\_type == PNG\_COLOR\_TYPE\_RGB) row\_bytes = width \* 3; // 3 bytes per pixel for RGB**

**vector<png\_byte> temp\_row(row\_bytes); // Temporary storage for a row**

**for (int i = 0; i < height / 2; i++) {**

**memcpy(temp\_row.data(), row\_pointers[i], row\_bytes);**

**memcpy(row\_pointers[i], row\_pointers[height - 1 - i], row\_bytes);**

**memcpy(row\_pointers[height - 1 - i], temp\_row.data(), row\_bytes);**

**}**

**}**

**[1] row\_bytes 계산**

**int row\_bytes = width \* 4; // 4 bytes per pixel for RGBA**

**if (color\_type == PNG\_COLOR\_TYPE\_RGB) row\_bytes = width \* 3; // 3 bytes per pixel for RGB**

**이미지의 한 픽셀이 RGBA 형식으로 저장돼 있을 때, 한 행의 바이트 수를 계산한다. 각 픽셀은 4바이트로 구성된다.**

**만약 color\_type이 PNG\_COLOR\_TYPE\_RGB일 경우, row\_bytes를 width \* 3으로 설정한다. 이 때는 RGB 형식의 이미지이므로 각 픽셀이 3바이트로 구성된다.**

**[2] 임시 행 데이터 저장**

**vector<png\_byte> temp\_row(row\_bytes);**

**한 행의 데이터를 임시로 저장하는데 사용할 temp\_row 벡터를 생성한다.**

**[3] 이미지 반전**

**for (int i = 0; i < height / 2; i++) {**

**이미지의 상반부와 하반부를 교환하는 데 반복문을 사용한다. height / 2만큼 반복하면 이미지의 중간까지 반전되는데, 이미지의 중간 지점까지 해당 과정을 반복하면 전체 image가 수직으로 뒤집힌다.**

**memcpy(temp\_row.data(), row\_pointers[i], row\_bytes);**

**현재 행의 데이터를 temp\_row에 복사한다.**

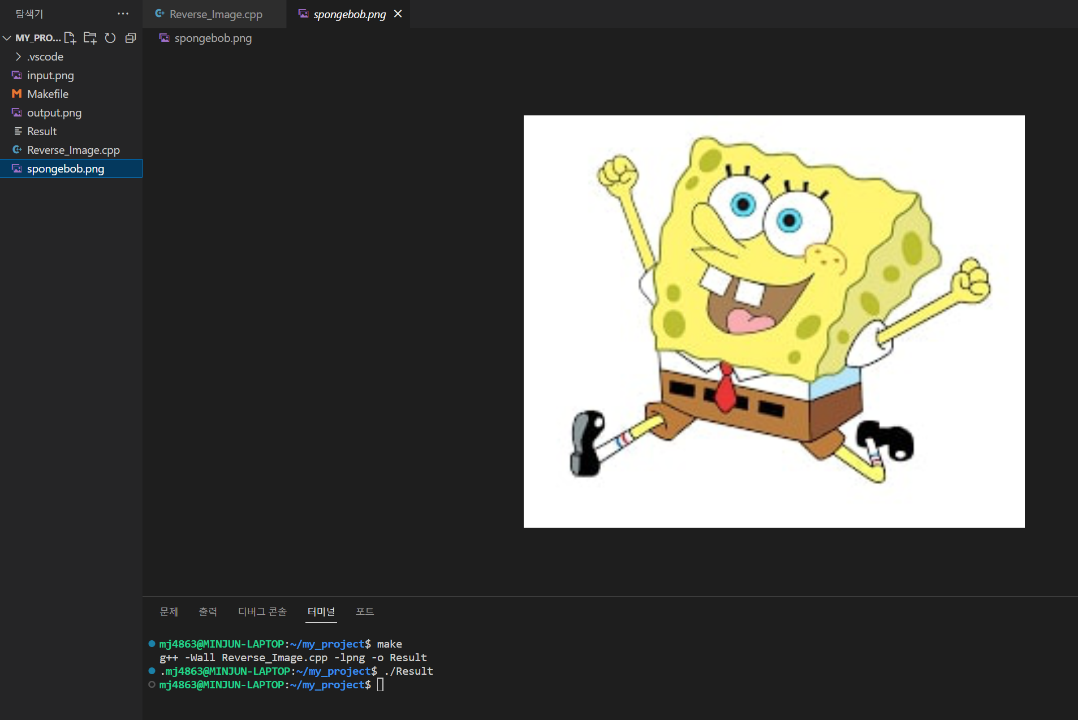
**memcpy(row\_pointers[i], row\_pointers[height - 1 - i], row\_bytes);**

**현재 행에 이미지의 아래쪽 행의 데이터를 복사한다. 즉, 현재 행의 데이터를 이미지의 위쪽으로 옮긴다.**

**memcpy(row\_pointers[height - 1 - i], temp\_row.data(), row\_bytes);**

**image의 아래쪽 행에 temp\_row에 저장된 데이터를 복사한다. 즉, imgae의 위쪽 행의 데이터를 이미지의 아래쪽으로 옮긴다.**

**flip\_image 함수를 사용하면 모든 PNG 이미지가 수직 방향으로 뒤집힌다!**

****

**스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**input.png 파일이 아닌 다른 png 파일을 이용하더라도 잘 뒤집히는 것을 확인할 수 있다!**