CSED 232 Object-Oriented Programing (Spring 2023)

Assignment # 3

- Inheritance & polymorphism -

Due date : 4월 28일 23시 59분

담당 조교 : 김건웅 (k2woong92@postech.ac.kr)

주의 사항

- 클래스 선언 및 정의를 main.cpp 파일에 작성하는 것을 금지합니다. 그 외에 선언 및 정의의 위치에 대한 제약은 없습니다.
- STL 사용이 가능합니다.
- 모든 C++ 문법이 사용 가능합니다.
- 문제에서 제공한 형식을 준수해야 합니다.
- 문제에 명시되어 있지 않더라도 소멸자(Destructor)와 같은 메모리 누수 방지를 위해 필요한 멤버함수는 필수적으로 구현되어야 합니다.
- 문제 조건이 복잡합니다. 모든 문제의 세부 조건을 꼼꼼히 읽어 보시기 바랍니다.
- 과제 관련 질문은 PLMS를 통해 문의 바랍니다.

감점

- 제출 기한에서 하루(24시간) 늦을 때마다 20%씩 감점
    - 1일(20%) , 2일(40%), … 5일(100%)
- 컴파일이 정상적으로 이루어지지 않을 경우 0점
- 제출 형식 위반 (파일 이름 등)

제출방식

채점은 Windows Visual Studio 2022(윈도우 사용자의 경우) 및 Ubuntu 20.04(lts)와 gcc version 9.4.0 (맥 사용자의 경우) 환경에서 이루어집니다. VS 로 작업했을 경우 작업하신 환경이 있는 visual studio 프로젝트 폴더에 Report 를 포함하여 zip 파일로 압축 후 제출해 주시기를 바랍니다. (x64 및 .vs 폴더는 전부 지워주십시오) 마찬가지로, 맥 이용자의 경우 소스 코드, 보고서, Makefile을 포함한 폴더를 압축해서 제출해주시면 됩니다. 폴더명은 '학번'으로 만들어 주시고, Report는 docx 나 pdf 형식으로 제출해주세요. 반드시 PLMS 를 통해 제출해주시기를 바랍니다. 이메일 제출은 인정되지 않습니다. 폴더 이름과 압축파일은 "Assign3_학번"으로 만드시면 됩니다 (e.g., Assign3_20229999).

공통 채점 기준

1. 프로그램 기능

- 프로그램이 요구 사항을 모두 만족하면서 올바로 실행되는가?

2. 프로그램 설계 및 구현

- 요구 사항을 만족하기 위한 변수 및 알고리즘 설계가 잘 되었는가?
- 문제에서 제시된 세부 조건을 모두 만족하였는가?
- 설계된 내용이 요구된 언어를 이용하여 적절히 구현되었는가?

3. 프로그램 가독성

- 프로그램이 읽기 쉽고 이해하기 쉽게 작성되었는가?
- 변수명이 무엇을 의미하는지 이해하기 쉬운가?
- 프로그램의 소스 코드를 이해하기 쉽도록 주석을 잘 붙였는가?

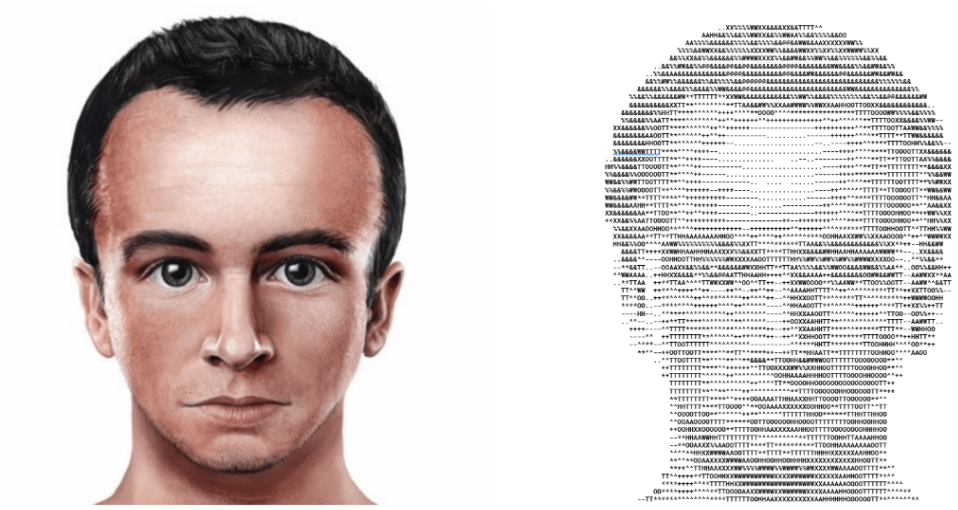
4. 보고서 구성 및 내용, 양식

- 보고서는 적절한 내용으로 이해하기 쉽고 보기 좋게 잘 작성되었는가?
- 보고서의 양식을 잘 따랐는가?

다른 사람의 프로그램이나 인터넷에 있는 프로그램을 복사(copy)하거나 간단히 수정해서 제출하면 학점은 무조건 'F'가 됩니다. 이러한 부정행위가 발견되면 학과에서 정한 기준에 따라 추가적인 불이익이 있을 수 있습니다.

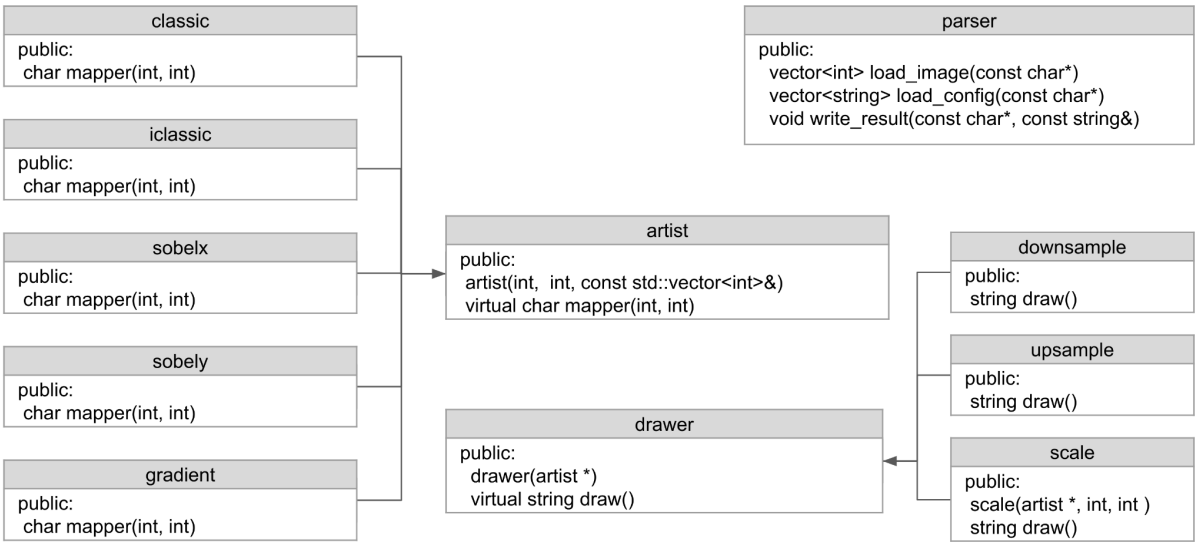
ASCII Art

ASCII art는 ASCII 문자만을 사용하여 그림이나 도형을 만드는 예술 형식이다. 예를 들어서, 그림 1과 같이 임의의 사진에서 어두운 부분에 해당하는 위치에 복잡한 글자를 위치시키고 밝은 부분에는 단순한 글자를 위치시키면 문자로만 이뤄진 그림이 아래와 같이 보이게 된다. 본 과제에서는 클래스 상속과 다형성을 사용해서 간단한 ASCII Art를 생성하는 프로그램을 만드는 것을 목적으로 한다.



[그림1. ASCII art example]

문제 해결을 위해 사용되는 class와 상속 관계는 아래 그림 2과 같다. 화살표는 상속을 의미하고 방향표시가 있는 class가 부모 class다. 예를 들어서 classic class는 artist class를 상속받는다.



[그림2. Class diagram]

프로그램을 구현에 아래와 같은 제약사항들이 따른다.

- 그림2에 명시된 멤버변수 및 함수는 반드시 구현되고 사용되어야 한다.
- 그림2에 명시된 멤버함수의 인자와 return type이 준수되어야 한다.
- 그림2에 명시된 class 외에 다른 class를 정의해서 사용할 수 없다.
- 제공된 함수(function) 외에 함수를 정의해서 사용할 수 없다.
- class diagram에 명시된 멤버함수(method)와 멤버변수 외에도 필요한 멤버함수 및 멤버변수를 자유롭게 추가할 수 있다.
- 주어진 **main** 함수는 절대로 수정할 수 없다**.**

그림 2에 표현된 클래스 및 main 함수에 대한 자세한 설명은 아래와 같다.

int main(int argc, char *argv[])

C++ main 함수에 int 와 char*[] 를 argument로 정의한 경우, 첫 번째 인자는 프로그램 실행시 입력받은 argument의 수, 두 번째 인자는 실제 argument 값들을 갖는다. 예를 들어서 임의의 compile된 프로그램 *hello* 를

./hello 2 world 1234

라는 명령어로 실행 시킬 경우 argc 는 4, argv[1] 는 2 argv[2] 는 world, argv[3] 는 1234가 된다. 참고로 argv[0]에는 실행파일경로가 자동으로 할당되며, 따라서 argc 는 3이 아닌 4가 된다.

본 과제에서 최종 compile된 프로그램은 실행 시 3개의 인자를 받는다. 첫 번째 인자는 input파일의 경로, 두 번째 인자는 configuration 파일의 경로, 세 번째 인자는 output 파일의 경로다.

주어진 main함수는 아래와 같으며 절대 수정을 불가한다.

```
int main(int argc, char *argv[]) {

  if (argc != 4) {
    cout << "argc is not 4, but " << argc << endl;
    throw;
  }

  // CREATE PARSER
  parser p;

  // LOAD IMAGE AND CONFIG
  vector<int> tokens = p.load_image(argv[1]);
  vector<string> configs = p.load_config(argv[2]);
```

```
  string style_target = configs[0];
  string drawer_target = configs[1];
  char *path_output = argv[3];

  int width = tokens[0];
  int height = tokens[1];
  vector<int> vals = {tokens.begin() + 2, tokens.end()};

  // CREATE ARTIST
  artist *style;
  if (style_target == "classic") {
    style = new classic(width, height, vals);
  } else if (style_target == "iclassic") {
    style = new iclassic(width, height, vals);
  } else if (style_target == "sobelx") {
    style = new sobelx(width, height, vals);
  } else if (style_target == "sobely") {
    style = new sobely(width, height, vals);
  } else if (style_target == "gradient") {
    style = new gradient(width, height, vals);
  } else {
    throw;
  }

  // CREATE DRAWER
  drawer *d;
  if (drawer_target == "drawer") {
    d = new drawer(style);
  } else if (drawer_target == "upsample") {
    d = new upsample(style);
  } else if (drawer_target == "downsample") {
    d = new downsample(style);
  } else if (drawer_target == "scale") {
    int scale_x = stoi(configs[2]);
    int scale_y = stoi(configs[3]);
    d = new scale(style, scale_x, scale_y);
  } else {
    throw;
  }

  // PERFORM DRAWING
  string output = d->draw();
  cout << output;

  // WRITE OUTPUT
  p.write_result(path_output, output);

  return 0;
}
```

vector<int> parser::load_image(const char*)

인자로 input file의 경로를 받고 해당 파일의 정보를 불러온다. input 파일의 내용은 일련의 숫자들이 delimiter '|' 을 통해 나눠져 있다. 예를 들어서 input 파일의 내용이

3|2|101|102|103|104|105|106

와 같다면, 해당 함수의 return 값은

vector<int> { 3 ,2, 101, 102, 103, 104, 105, 106 }

이 된다. 여기서 처음 두 값은 그려질 그림의 width와 height를 의미하고 나머지 값은 각 좌표에 해당되는 pixel 값이다. 예를 들어서 위 정보는

101 (0,0)	102 (1,0)	103 (2,0)
104 (0,1)	105 (1,1)	106 (2,1)

와 같이 시각화 될 수 있다. 여기서 괄호 안의 정보는 x,y 좌표를 의미한다. 이후 artist와 drawer class를 통해서 해당 값이 특정 ASCII 문자로 변환되며 그림을 그리게 된다.

vector<string> parser::load_config(const char*)

인자로 config file의 경로를 받고 해당 정보를 불러온다. config 파일의 내용은 일련의 text들이 delimiter '|' 을 통해 나눠져 있다. 첫 번째 값은 사용될 artist class의 이름, 두 번째 값은 사용될 drawer class의 이름을 의미한다. 예를 들어서 config 파일의 내용이

iclassic|upsample

와 같다면, 해당 함수의 return 값은

vector<string> { "iclassic", "upsample"}

이 된다.

void parser::write_result(const char*, const string&)

첫 번째 인자로 저장할 파일 경로, 두 번째 인자로 저장할 내용이 들어온다. 예를 들어서

p.write_result("output.txt","hello world")

를 호출하면 output.txt파일이 "hello world"라는 내용이 담겨서 생성된다. 해당 함수로 최종 결과를 저장하게 된다. 예시는 아래와 같으며 예시를 그대로 사용해도 무방하다.

```
void parser::write_result(const char *path, const string& contents) {
  ofstream myfile;
  myfile.open(path);
  myfile << contents;
  myfile.close();
}
```

artist(int, int, const std::vector<int>&)

artist class는 그림의 스타일을 정의하는 class다. 다양한 스타일 class들이 artist class를 상속받아서 자신만의 스타일을 정의하게 될 것이다. 그리고 이후 설명할 drawer class가 artist class의 instance를 인자로 받아서 최종적인 그림을 그리게 된다.

생성자는 첫 번째 인자로 그림의 width, 두 번째 인자로 그림의 height, 세 번째 인자로 그림의 각 좌표에 해당하는 pixel 값을 받는다. 즉, input 파일 parsing 결과를 인자로 받는다.

virtual char mapper(int, int)

첫 번째 인자로 x coordinate, 두 번째 인자로 y coordinate을 받고, 사용될 ASCII 문자를 return한다. 다형성을 구현하기 위한 virtual method이기 때문에 해당 class에서는 구현되지 않고 상속 받는 class에서 구현된다.

char classic::mapper(int, int)

classic class의 mapper는 가장 일반적인 ASCII Art의 mapping을 구현한다. 이미지는 display에 표현될 때 pixel값이 작을수록 어둡게 나타난다. 이 관찰에 근거해서 pixel 값이 낮을수록 시각적으로 dense한 ASCII 문자를 대입시켜 그림을 표현한다. 구체적으로 classic::mapper는 pixel을 표현하기 위해서 15개의 ASCII 문자

‘@’ ‘&’ ‘%’ ‘W’ ‘X’ ‘A’ ‘H’ ‘O’ ‘T’ ‘*’ ‘^’ ‘+’ ‘-’ ‘.’ ‘ ‘

를 사용한다. 왼쪽부터 pixel 값이 낮은 경우에 대응되고 각 문자는 [0, 254] 사이의 값을 17개씩 균일하게 커버한다. 예를 들어서 ‘@’는 [0, 16], ‘&’ 는 [17, 33]를 커버한다. 예외적으로 마지막 문자만 18개, [238, 255]의 범위를 커버한다. 각 pixel이 ASCII 문자로 변하는 예시는

2	18
18	37

@	&
&	%

와 같다. 참고로 그림1의 ASCII art는 classic::mapper를 통해 그려진 결과다.

char iclassic::mapper(int, int)

iclassic은 inverted classic의 약자다. iclassic::mapper는 사용되는 ASCII문자의 순서를 반대로 사용한 것 외에 classic::mapper와 모든 logic이 동일하다. 예를 들어서 ' '가 [0, 16]을 커버하고 '@'가 [238, 255]를 커버한다.

char sobelx::mapper(int, int)

sobel operation은 영상에서 edge성분을 검출할때 사용되는 연산이다. 본 과제에서 sobelx::mapper는 임의의 pixel에서 x축 양의 방향으로 인접한 pixel과의 차이가 50이상인 경우 '|'를, 그렇지 않은 경우 공백문자 ' '를 리턴한다. 예를 들어서

100	160	120
100	30	100
100	100	170

|		
|	|	
	|	

와 같이 왼쪽의 pixel 값에 대해서 오른쪽 같은 ASCII 문자가 그려진다.

char sobely::mapper(int, int)

sobelx에서 사용된 logic과 두 가지 세부사항이 다르다. 먼저 x축이 아닌 y축의 방향을 고려한다. 그리고 리턴되는 문자가 '|'가 아닌 '-'이다.

char gradient::mapper(int, int)

gradient::mapper는 sobelx::mapper와 sobely::mapper의 logic을 그대로 반영한다. 추가적으로 x,y축 양의 방향 모두 인접한 pixel과의 차이가 50이상인 경우 '+'를 리턴한다. 예를 들어서

100	160	120
100	30	100
100	100	170

|	-	
|	+	-
	|	

와 같이 왼쪽의 pixel 값이 오른쪽과 같은 ASCII 문자로 그려진다.

drawer(artist *)

drawer class는 artist가 갖는 method를 사용해서 최종적으로 그려지는 string을 만들어낸다. 따라서 drawer 생성자는 첫 번째 인자로 artist pointer instance를 받는다.

string drawer::draw()

최종 ASCII 문자로 구성된 그림을 하나의 string으로 리턴한다.

string downsample::draw()

upsample과 downsample은 이미지 크기를 키우고 줄이는데 사용되는 방법이다. 본 과제에서는 Nearest-neighbor 기반의 upsample과 downsample을 downsample::draw와 upsample::draw를 통해서 모방한다.

downsample::draw()는 drawer::draw()로 그려지는 그림의 크기를 절반으로 줄인다. 구체적으로 좌표 (0, 0)은 반드시 그려지고, 해당 좌표를 기준으로 홀수 값을 갖는 좌표의 문자들은 무시된다. 예시는

@	&	**%**	&
&	%	&	%
&	%	**&**	%
&	%	&	%

@	%
&	&

와 같으며 왼쪽이 원본이고 오른쪽이 downsample 결과다.

string upsample::draw()

downsample::draw()와 반대로 그림의 크기를 두 배로 키운다. 키우는 과정에서 각 문자를 반복한다. 예시는

@	%
&	&

@	@	%	%
@	@	%	%
&	&	&	&
&	&	&	&

와 같으며 왼쪽이 원본이고 오른쪽이 upsapmle 결과다.

scale(artist *, int, int)

scale class의 생성자는 예외적으로 artist*외에 추가적인 두 개의 int 인자를 받는다. 두 int 인자는 각각 x, y축으로 확장 및 축소될 값을 의미한다. 이 경우 예외적으로 config 파일에 2개가 아닌 4개의 정보가 포함되어야 하며 3,4 번째 정보가 해당 scale 생성자의 인자로 들어간다. 해당 내용은 main 함수에 이미 작성 되어있다.

string scale::draw()

downsample::draw()와 upsample::draw()이 축을 구분하지 않고 단순히 2배 확장 및 축소를 했다면, scale::draw()는 각 축에 대해서 정해진 배율의 확장 축소를 수행한다. 배율이 자연수인 경우 해당 값의 배율로 그림을 확장한다. 배율이 음의 정수인 경우 해당 값의 음의 역수를 배율로 사용한다. 마지막으로 0 인 경우는 허용하지 않는다. 예를 들어서 x축과 y축 배율이 3, -2 인 경우 x축으로 3배, y 축으로 1/2배 확장 및 축소한다.

아래 이어지는 예시는 프로그램의 **input**과 이에 대응되는 **output**예시들이다**.**

```
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
@@@@%%%WWXXXHHHOOTTT***++---....
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
....---++***TTTOOHHHXXXWW%%%@@@@
```

output using input1.txt and confit1.txt

output using input2.txt and confit2.txt

최종적으로 풀어야하는 문제는 아래와 같다**.** 문제 **1,2,3**은 코드 구현 문제로 **test case**를 통해서 채점되고**,** 문제 **4,5**는 보고서에 작성한다**.**

문제1> parser 클래스를 구현하라

문제2> artist class 및 artist를 상속받는 class 그리고 drawer class를 구현하라

문제3> drawer class를 상속받는 class들을 구현하라

문제4> drawer 생성자가 artist가 아닌 artist*를 인자로 받는 이유를 설명하라 (보고서에 작성)

문제5> artist 클래스를 상속받아서 자신만의 style을 정의하고 그림을 출력하라 (보고서에 작성)

문제6> 제시된 클래스들을 객체 지향 프로그래밍 관점에서 개선할 수 있다면 어떤 점을 더 개선할 수 있을지 토의하라 (보고서에 작성)