2023 Spring OOP Assignment Report

과제 번호 : 3
학번 : 20190445
이름 : 허수범
Povis ID : sbh408

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.
I completed this programming task without the improper help of others.

프로그램을 하다 보면 결정해야 할 세부 사항이 많은데, 이러한 세부 사항을 처리한 방법과 이유를 보고서에 쓰십시오.
독창적인 아이디어와 추가 기능은 보너스 점수를 받을 수 있으므로, 보고서에 명확히 기재하십시오.
문제가 여러 개인 경우, 각 문제별로 정리해서 작성합니다.

아래 문항별 설명은 편의를 위한 것으로, 삭제하고 제출한다.

1. 프로그램 개요

본 프로그램은 command line argument로 input, configuration, output 파일의 경로를 받아 두 파일의 요구사항에 맞게 ASCII art를 그리는 프로그램이다. Input file에는 원하는 이미지의 크기(height와 width)와 각 좌표의 픽셀 값이 일련의 숫자들로 '|'로 나뉘어진 채로 작성되어 있다. Configuration에는 사용될 artist class의 이름과 사용될 drawer class의 이름, 그리고 기타 configuration이 일련의 text들로 '|'로 나뉜 채로 작성되어 있다. 결과는 지정해준 output 파일에 저장되며, console에서도 확인할 수 있다.

Input과 configuration은 이후 설명할 parser class의 함수들로 읽어들인다.

프로그램 디렉토리 내에는 class들의 선언이 작성되어 있는 헤더 파일 header.hpp, 멤버 함수들과 main함수의 정의가 작성되어 있는 소스 코드 파일 main.cpp, Makefile이 있다.

2. 프로그램의 구조 및 알고리즘

매뉴얼에 주어진 class만을 선언하고 정의하였다.

parser class는 세가지 멤버함수를 가지고 있다.

vector<int> parser::load_image(const char* input): 해당 함수는 input 파일의 경로를 문자열로 받아 delimiter '|'로 파일 내의 각 숫자를 쪼개 하나의 정수 벡터로 return한다. 먼저

ifstream class 생성자를 이용해 ifstream 인스턴스 생성과 input file open을 동시에 진행하고, 파일 열기가 성공적일 때, input파일의 내용을 모두 동적으로 할당된 하나의 문자열에 저장한 후, 해당 문자열을 sstream library의 객체 stringstream과 getline을 이용해 delimiter '|'로 쪼개 정수로 변환하여 push_back()으로 정수 벡터에 저장하고, 메모리 할당 해제 후 결과 벡터를 return한다. char divider는 delimiter '|'를 저장하는 문자 변수, vector<int> loaded는 return할 정수 벡터, ifstream fileIn은 파일 입력을 위해 선언한 인스턴스, stirng* line은 inpur file의 전체 내용을 받을 문자열을 동적 할단한 포인터, string value는 delimiter로 쪼개진 문자열을 잠시 저장할 인스턴스, stringstream ss는 문자열을 delimiter를 제외하고 쪼개기 위해 선언한 객체이다.

vector<string> parser::load_config(const char* input): 해당 함수는 위의 함수와 매우 비슷하다. char divider, ifstream fileIn, string* line, string value, stringstream ss는 모두 동일한 역할을 하지만, 쪼개진 문자열을 전수로 변환하지 않는다는 점과, 결과를 저장하고 return되는 벡터 vector<char> loaded가 character 벡터라는 점이 다르다.

void parser::write_result(const char* path, const string& contents): 해당 함수는 파일을 저장할 경로와 저장할 내용이 들은 문자열을 각각 argument로 받는다. 저장할 내용이 들은 문자열의 내용을 지정된 경로에 저장한다. 매뉴얼의 함수를 그대로 사용하였다.

artist class는 세 개의 멤버 변수와 생성자와 소멸자를 포함하여 7개의 멤버 함수를 가진다. 먼저 멤버 변수 int width와 height는 그림의 너비와 높이가 저장되고, vector< vector<int> > pixels는 ascii art로 변환되기 전의 정수로 이루어진 그림이 저장되는 2차원 벡터이다. 세 맴버 변수 만이 private member로 선언되었고, 멤버함수들은 모두 public member이다ㅏ.

char artist::clssc(int): 이 함수는 이후 classic과 iclassic에서 사용된다. 입력으로 정수를 받고, switch문을 통해 해당 정수에 해당되는 character 출력을 return한다. 낮은 정수값이 입력될수록 더욱 dense한 문자를 return하도록 한다.

int artist::getPixel(int x, int y), int artist::getWidth(), int artist::getHeight(): 모두 헤더 파일에 인라인 함수로 선언되고 정의되었다. 먼저 int getPixel()은 입력으로 받은 좌표의 값을 반환해주고, getWidth()는 그림의 너비, getHeight()는 그림의 높이를 반환해준다.

artist::artist(int, int, const vector<int>&): 이는 artist class 생성자이다. 첫 번째 argument는 width에, 두 번째 argument는 height에 assignment 해주고, 세 번째 인자인 1차원 정수 벡터는 for문과 push_back()을 이용해 pixels에 2차원 벡터로 다시 저장해 준다. 1차원 벡터를 각 row로 쪼개서 pixels에 push해주기 위해 각 row를 저장할 정수 벡터 row를 선언하여 사용하였다.

virtual char artist::mapper(int, int), virtual artist::~artist(): 먼저 mapper는 artist자체적으로는 없고, artist를 상속받는 객체들에서만 정의되므로 pure virtual 멤버 함수가 되도록 "= 0"를 붙여주었다. artist 소멸자의 경우에는 자식 class들에서의 혼동을 막기 위해 virtual로 선언해주었지만, 따로 기능을 하지는 않아 특별한 정의는 해주지 않았다.

이후 artisti를 상속받는 클래스들의 생성자는 initializer list를 이용해 artist 생성자를 호출한다. 문제 5에서 새로 만든 클래스 avgfilter와 bilatfiler 클래스를 제외하고는 생성자들이 다른 기능을 하지 않는다.

밑에 설명할 classic, iclassic, sobelx, sobely의 경우, artist를 public 상속받으며, public 멤버 함수 mapper와 클래스 생성자와 소멸자를 가진다. 생성자는 위에서 언급했다시피, artist의 생성자를 initializer list로 호출하는 것 외에는 별도의 기능을 하지 않고, 소멸자 역시 특별한 역할을 하지 않도록 했다.

char classic::mapper(int xCoord, int yCoord): 가장 일반적인 ascii art의 mapping을 구현하는 함수이다. 입력받은 x, y좌표를 이용해 getPixel함수로 해당 좌표의 값을 받고, 이를 17로 나눈 후 위에서 설명한 artist::clssc()함수에 대입한 값을 return한다.

char iclassic::mapper(int xCoord, int yCoord): ascii 문자의 순서를 반대로 mapping하는 함수로, 동일하게 clssc()함수를 호출하지만, 14에서 해당 픽셀의 값을 17로 나눈 해를 뺀 값을 대입하며, if문을 활용해 픽셀의 값이 255인 경우에는 '@'를 리턴하도록 하였다.

char sobelx::mapper(int xCoord, int yCoord), char sobelx::mapper(int xCoord, int yCoord): 두 함수는 x 방향과 y 방향 sobel필터를 구현하는 함수다. If else문을 활용해 좌표가 경계의 값일 경우 빈 칸을 리턴하도록 하였고, 그렇지 않다면 sobelx의 경우 오른쪽 픽셀의 값, sobely의 경우 아래의 픽셀의 값과 해당 좌표의 값을 비교하여 값이 50 이상 차이가 나면 각각 '|'와 '-'를 return하도록 하였으며, 나머지 경우에는 모두 빈 칸을 return하도록 하였다.

char gradient::mapper(int xCoord, int yCoord): 해당 함수는 sobelx와 sobely를 합쳐 두 필터에 모두 걸리는 값이면 '+'를 리턴하도록 하는 함수이다. 먼저 각각 x축 방향과 y축 방향 인접 픽셀과의 값 차이를 저장하기 위해 xDiff와 yDiff 정수 변수를 선언하고, if else 문을 활용해 입력받은 좌표 인자가 경계값이 아닐 경우 차이값을 xDiff와 yDiff에 저장하고, 경계값일 경우 0을 저장하도록 하였다. 이후 또 if else문을 활용해 각 경우에 따라 '|', '-', '+', ' '를 리턴하도록 하였다.

drawer class는 private멤버 변수로 생성자에서 입력받을 artist 포인터 myArtist를 갖고,

public 멤버로 생성자와 소멸자, 그리고 네개의 멤버 함수를 가진다.

char getDPixel(int x, int y), int getDWidth(), getDHeight(): getDPixel에서는 myArtist의 mapper에 접근하여 입력받은 좌표의 문자를 return한다. getDWidth()와 getDHeight()의 경우에는 myArtist의 getWidth()와 getDHeight()에 접근하여 그림의 너비와 높이를 return한다. 셋 다 헤더 파일에 정의된 인라인 함수이다.

생성자: 단순히 인자로 받은 artist 포인터를 myArtist에 assignment해준다.

virtual string draw(): 이후 override될 것이기 때문에 virtual이 붙었지만, 순수 drawer도 기능을 해야 하기 때문에 정의되었다. myArtist에 해당하는 그림을 하나의 문자열(string)으로 반환하는 함수인데, 결과 string restult을 먼저 선언해 주고 ""로 초기화해주며, 이중 for문을 바깥 loop는 높이만큼, 안쪽 loop는 너비만큼 loop하도록 설정하여 모든 문자를 getDPixel(x, y)로 접근해주어 string에 append해주도록 하였고, 안쪽 loop 바깥에 개행문자를 append하도록 하여 string을 출력할 때 원하는 크기의 그림이 나올 수 있도록 하였다.

소멸자: 헤더파일에 정의된 인라인 함수로, myArtist를 delete해준다. 즉, drawer가 delete 될 때 해당되는 artist 객체 역시 할당 해제 된다.

밑에 설명할 downsample과 upsample class의 경우 public 멤버로 각각 draw()와 생성자와 소멸자를 가진다. 하지만 생성자는 initializer list로 drawer의 생성자를 호출하는 것 이외에는 일을 하지 않고, 소멸자 역시 특별한 기능을 하지 않도록 하였다.

string downsample::draw(), string upsample::draw(): 둘은 사실상 위의 draw()와 동일한 구조를 가지고 있다. string result를 선언하고 빈 문자열로 초기화해준 후 달라지는 점은 for loop의 iteration 횟수와 접근하여 append하는 픽셀 문자의 값이다. 먼저 downsample의 경우 바깥 loop는 높이의 반만큼, 안쪽 loop는 너비의 반만큼 iterate하도록 하였고, upsample은 반대로 각각 두배 iterate하도록 하였다. result 문자열에 append하기위해 접근하는 getDPixel의 좌표는 downsample의 경우 j * 2, i *2, upsample의 경우 j / 2, i / 2가 되도록 하였다. j는 안쪽 for loop의 iteration 문자, i의 바깥쪽 for loop의 iteration 문자이다. i와 j는 정수 변수이기 때문에 이런 방식으로 원하는 좌표의 픽셀에 접근할 수 있었다.

scale class의 경우, private 변수로 x축 scale값 xProp, y축 scale값 yProp을 가지며, 생성자, 소멸자, draw()를 public member로 갖는다.

생성자: 생성자는 세 개의 인자를 받는다 첫 번째 인자는 artist 포인터, 두번째와 세번째 인자는 각각 배율값으로 정수형 변수이다. 먼저 initializer list로 drawer의 생성자를 호출

하여 artist 포인터를 인자로 주고, 두번째와 세번째 인자를 각가 xProp과 yProp에 assign 한다.

string scale::draw(): xProp과 yProp의 배율로 augment된 이미지를 string으로 리턴하는 함수이다. 먼저 return할 string result를 선언하고 초기화해준 후, 실제 배율이 될 X와 Y를 double형 변수로 선언해주고, 각각에 일단은 xProp과 yProp을 assign해준다. 이후 if문을 이용해 각각 xProp이 음수이거나 yProp이 음수이면, X = -1.0 / X, Y = Y -1.0 / y가 되도록 하였다. 음수인 경우애는 축소 배율이 되어야 하기 때문이다. 이후는 upsample혹은 downsaple과 사실상 동일하다. 하지만 outer for loop의 iteration 횟수는 height * Y, inner for loop는 width * X가 되도록 하고, append하기 위해 접근하는 pixel의 좌표는 int(j / X), int(i / Y)가 되도록 하였다.

Main함수의 경우 새로 만든 style을 test할 때 외에는 수정하지 않았으므로 설명하지 않는다.

3. 토론 및 개선

처음으로 클래스의 상속을 다루어 보았기 때문에 헷갈리는 점들이 많았다. 특히 virtual 함수의 사용이나 생성자, 소멸자 등의 선언과 정의, private 멤버 접근 가능 여부 등 생소한 개념들이 많이 사용되어 힘든 점이 있었다. 하지만, 적절한 가이드라인과 자유도로 해당 내용들을 마음껏 연습해볼 수 있어 개념 습득이 쉬웠다.

문제 4>

drawer 생성자가 artist가 아닌 artist*를 인자로 받는 이유는 크게 두가지로 해석했다.

첫번째는 code redundancy 때문이다. main함수에서는 config 에 따라 artist 자식 클래스 객체를 먼저 동적 할당한 후, config에 따른 drawer나 drawer 자식 클래스 객체를 동적할당하도록 되어있다. 이는 코드 redundancy를 줄이기 위한 의도로 해석할 수 있다.

Artist 객체의 핵심 멤버 함수는 mapper함수이다. 허나 이 함수는 어떤 자식 클래스냐에 따라 매우 다른 기능을 하게 된다. 따라서 각 자식에서 override할 수 있도록 mapper는 artist에서 virtual함수로 선언되어 있다. 이렇게 구성을 한 이유는 drawer 객체를 생성할 때, 어떤 artist 자식 클래스냐에 따라 다르게 drawer를 생성할 필요 없이, 단순히 artist 포인터로 drawer 객체를 생성하고, drawer 객체 내에서 draw를 할 때에 단순히 mapper로 접근해도, 해당 artist 객체가 어떤 종류인지에 따라 적절한 mapper 함수가 호출되게 하기 위해서이다.

두번째는 효율적 메모리 사용을 위해서이다. artist객체들은 2차원 벡터를 멤버 변수로 가지기 때문에, 메모리를 많이 차지할 수 있다. 이를 call by value를 하면, 불필요한 memory copy가 발생할 수 있다. 이를 방지하기 위해 call by reference를 하도록 artist 포

인터를 인자로 받는 것이다.

문제 5>

```
class avgfilter : public artist{
    private:
        int fsize;
    public:
        char mapper(int, int);
        avgfilter(int Width, int Height, const vector<int>& loaded,
int size) : artist(Width, Height, loaded) {fsize = size;}
        ~avgfilter() {}
};

class bilatfilter : public artist{
    private:
        int fsize, sSigma, iSigma;
    public:
        char mapper(int, int);
        bilatfilter(int Width, int Height, const vector<int>& loaded
, int size, int s, int i) : artist(Width, Height, loaded) {
            fsize = size;
            sSigma = s;
            iSigma = i;
        }
        ~bilatfilter() {}
};
```

```
//mapper of average filter
char avgfilter::mapper(int xCoord, int yCoord){
    int radius = fsize / 2;
    int startx, endx, starty, endy;
    if(xCoord - radius < 0)
        startx = 0;
    else
        startx = xCoord - radius;

    if(yCoord - radius < 0)
        starty = 0;
    else
        starty = yCoord - radius;

    if(xCoord + radius > getWidth() - 1)
        endx = getWidth() - 1;
    else
        endx = xCoord + radius;

    if(yCoord + radius > getHeight() - 1)
        endy = getHeight() - 1;
    else
        endy = yCoord + radius;

    int sum = 0;
    int count = 0;
    for(int i = starty; i <= endy; i++){
        for(int j = startx; j<= endx; j++){
            sum += getPixel(j, i);
            count++;
        }
    }

    int average = 0;
    if(count > 0)
        average = sum / count;

    return clssc(average / 17);
}
```

```
//mapper of bilateral filter
char bilatfilter::mapper(int xCoord, int yCoord){
    int radius = fsize / 2;
    int startx, endx, starty, endy;
    if(xCoord - radius < 0)
        startx = 0;
    else
        startx = xCoord - radius;

    if(yCoord - radius < 0)
        starty = 0;
    else
        starty = yCoord - radius;

    if(xCoord + radius > getWidth() - 1)
        endx = getWidth() - 1;
    else
        endx = xCoord + radius;

    if(yCoord + radius > getHeight() - 1)
        endy = getHeight() - 1;
    else
        endy = yCoord + radius;

    double sWeight = 1.0 /  (2.0 * pow(sSigma, 2));
    double iWeight;
    double sum = 0;
    double wSum = 0;
    double sDis, sWeightVal, iDis, weight;
    for(int i = starty; i <= endy; i++){
        for(int j = startx; j<= endx; j++){
            sDis = sqrt(pow(j - xCoord, 2) + pow(i - yCoord, 2));
            sWeightVal = exp(- pow(sDis, 2) * sWeight);
            iDis = abs(getPixel(j, i) - getPixel(xCoord, yCoord));
            iWeight = exp(- pow(iDis, 2) / (2.0 * pow(iSigma, 2)));
            weight = sWeightVal * iWeight;
            sum += weight * getPixel(j, i);
            wSum += weight;
        }
    }

    return clssc(int(round(sum / wSum)) / 17);
}
```

새 style을 정의하기 위해 작성한 코드. 구현한 average filter는 config에서 average filter의 크기를 받도록 하였고, 해당 크기만큼의 인접 픽셀들의 값을 평균내어 mapping하도록 하였다. 아래 사진들은 각각 classic으로 출력했을 경우, average filter size가 5일때, average filter size가 10일때 이다. 점점 경계가 흐릿해지는 것을 볼 수 있다. 원래 average filter의 의미는 noise가 있을 때인데, 원본 input에 노이즈가 없어 효과를 확인하기 어렵다.

추가로 bilateral filter역시 구현해 보았다. 아래 참고 문서 등을 참고하여 경계선을 유지한 채 흐릿해지는 필터를 구현해보려 하였으나, 원본 input에 노이즈가 없고, 화질이 애초에 낮았던 관계로 효과를 확실히 확인할 수는 없었다. Sigma값들을 추가적으로 입력 받고, average filter와 비슷한 알고리즘으로 연산을 수행하였다. 결과는 다음과 같다. 두 sigma 값과 filter size를 모두 10으로 했음에도 불구하고, 육안으로 효과를 확인하기 어려워 아쉬었다.

문제 6>

이번 과제에서 개선할 수 있는 점은 code redundancy라고 생각한다. 각각 artist와 drawer 자식 클래스들 멤버 함수들 간의 유사성이 매우 큰 경우가 있었다. 가령 sobelx와 sobely의 mapper 함수, downsample과 upsample의 draw함수들 간에는 큰 유사성이 있었다. Friend를 이용하거나 겹치는 class들은 합쳐 원할때 서로 다른 기능을 하게 하는 등의 방법을 이용해 redundancy를 줄일 수 있었을 것이라 생각한다.

4. 참고 문헌

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=won19600&logNo=221770735004 bilateral filteing을 구현하기 위해 참고한 문헌.