실습 9 - 10주차

학과 : 전자공학과 학번 : 2023104322 이름 : 현시온

- 과제는 pdf로 변환하여 제출(과제 문서 첫 줄에 학과/학번/이름 포함)
- 과제는 순서대로 작성하며, 문제와 설명을 모두 포함(형식이 맞지 않으면 감점)
- 프로그램을 작성하는 문제는 소스코드와 실행 결과를 모두 text로 붙여넣기(그림으로 포함하지 말 것)하고 코드 설명 및 결과에 대한 설명을 포함해야 함
- 문의 사항은 이메일(nize@khu.ac.kr) 또는 오픈 카톡방을 이용

1. 아래의 코드는 csv (comma-separated values) 파일을 읽어서 vector로 저장하는 함수이다. http://www.kaggle.com/saurabh00007/diabetescsv에서 diabetes.csv을 다운로드 받아서 동작을 확인하고 (main 함수 작성), 함수의 동작을 설명하라.

```
void ReadCsv(std::string FileName,
        std::vector<std::vector<std::string>> &Data) {
        std::ifstream ifs;
        ifs.open(FileName);
        if(!ifs.is_open()) return;

        std::string LineString = "";
        std::string Delimeter = ",";
        while(getline(ifs, LineString)) {
             std::vector<std::string> RowData;
             unsigned int nPos = 0, nFindPos;
             do {
              nFindPos = LineString.find(Delimeter, nPos);
              if(nFindPos == std::string::npos) nFindPos = LineString.length();

              RowData.push_back(LineString.substr(nPos, nFindPos-nPos));
              nPos = nFindPos+1;
             } while(nFindPos < LineString.length());
             Data.push_back(RowData);
        }

        ifs.close();
}
```

코드 및 설명:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>

void ReadCsv(std::string FileName,
    std::vector<std::vector<std::string>>& Data) {
    //문자열 FileName과 문자열 벡터의 벡터의 포인터 Data를 파라미터로 하는 함수 ReadCsv를 정의
    std::ifstream ifs;
    ifs.open(FileName);
    if (!ifs.is_open()) return;
    //파일 입력을 위한 객체 ifs를 선언하고 FileName을 인수로 하여 매소드 open을 호출. 즉 문자열
FileName에 해당하는 경로의 파일을 여는데, 만약 파일 열기에 실패했다면 함수는 즉시 종료한다.
    std::string LineString = "";
    std::string Delimeter = ",";
    //가공 전 파일의 한줄씩 저장할 문자열 LineString과 가공하면서 제거할 쉼표를 저장한
Delimeter 선언 및 초기화.
    while (getline(ifs, LineString)) {
        std::vector<std::string> RowData;
        unsigned int nPos = 0, nFindPos;
        //열기에 성공한 파일을 담고 있는 ifs에서 한줄씩 LineString에 저장하면서 이 과정을 모든
줄을 읽을때까지 반복. 그리고 ifs의 각 줄의 데이터를 저장할 문자열 벡터 RowData 선언.
        do {
            nFindPos = LineString.find(Delimeter, nPos);
            if (nFindPos == std::string::npos) nFindPos = LineString.length();

            RowData.push_back(LineString.substr(nPos, nFindPos - nPos));
            nPos = nFindPos + 1;
        } while (nFindPos < LineString.length());
        Data.push_back(RowData);
        //find 매소드를 통해 LineString의 인덱스 nPos, 즉 첫번째 문자부터 Delimeter, 즉 쉼표를
찾아 그 문자의 인덱스를 nFindPos에 할당. 만약 찾지 못해 std::string::npos가 할당된다면
nFindPos에는 LineString의 길이를 재할당.
        //substr 매소드를 통해 인덱스 nPos ~ 인덱스 (nFindPos - nPos) 만큼의 부분 문자열을
추출하여 매소드 push_back을 통해 RawData에 추가.
        //find 함수는 첫번째로 찾은 동일한 문자의 인덱스까지의 범위만 탐색하므로, 남은
LineString 부분을 탐색하기 위해 nPos를 nFindPos에 1을 더하고, do-while 반복문에 따라 아직
nFindPos가 LineString의 길이보다 작다면 다시 해당 과정을 반복.
    }

    ifs.close();
    //모든 과정을 마치고 열었던 파일을 닫는다.
}

int main() {
    std::string FileName = "C:₩₩Users₩₩samsung₩₩Downloads₩₩archive₩₩diabetes.csv";
    std::vector<std::vector<std::string>> Data;
    ReadCsv(FileName, Data);
    for (const auto& element : Data[1]) {
        std::cout << element << ", ";
    }
    std::cout << '₩n';
```

```
    return 0;
}
    //데이터를 추출하고 싶은 파일의 경로를 문자열로 하고, 데이터를 저장하고 싶은 2차원 벡터를
인수로 하여 함수를 호출하면 정상 작동한다.
    //우선 데이터의 첫번째 줄은 데이터 값의 이름이라 두번째 줄을 출력하도록 구성하였다.
```

2. 아래의 코드에서 IntPoint는 2차원 평면상의 점을 표현하는 클래스이며 x, y는 평면상의 좌표를 저장하는 멤버이며, Rectangle은 회전되지 않은 직사각형 표현하는 클래스로 왼쪽-코너점을 corner로, 폭과 높이를 각각 width와 height로 저장한다. 주석의 내용과 같이 동작하도록 코드를 완성하고 동작을 확인할 main 함수를 작성하라.

```
#include <iostream>
class IntPoint {
public:
      int x, y; // x, y 좌표
      IntPoint(int x, int y): x(x), y(y) {}
};
class Rectangle {
      IntPoint corner; // 직사각형의 왼쪽-아래 코너 점
      int width; // 직사각형의 폭
      int height; // 직시각형의 높이
public:
      Rectangle(IntPoint pt, int w, int h): corner(pt),
            width((w < 0) ? 0 : w), height((h < 0) ? 0 : h) {}
      int perimeter() {
            return 2*width + 2*height;
      }
      int area() {
            return width * height;
      }
      int get_width() {
           return width;
      }
      int get_height() {
            return height;
      }
      // 현재 인스턴스 사각형과 r이 겹쳐 있다면 true, 그렇지 않으며 false
      bool intersect(Rectangle r) {
           // 코드 작성
```

```
        }
        // 대각선의 길이(int 형)를 반환
        int diagonal() {
            // 코드 작성
        }
        // 사각형의 중심점의 좌표를 IntPoint 형으로 반환
        IntPoint center() {
            // 코드 작성
        }
       // 현재 인스턴스 사각형의 내부(경계포함)에 pt가 있으면 true,
       // 그렇지 않으면 false
        bool is_inside(IntPoint pt) {
            // 코드 작성
        }
};
```

코드 및 설명:

```
#include <iostream>
#include <cmath>
class IntPoint {
public:
    int x, y; // x, y 좌표
    IntPoint(int x, int y) : x(x), y(y) {}
};
class Rectangle {
    IntPoint corner; // 직사각형의 왼쪽-아래 코너 점
    int width; // 직사각형의 폭
    int height; // 직시각형의 높이
public:
    Rectangle(IntPoint pt, int w, int h) : corner(pt),
        width((w < 0) ? 0 : w), height((h < 0) ? 0 : h) {}
    int perimeter() {
        return 2 * width + 2 * height;
    }
    int area() {
        return width * height;
    }
    int get_width() {
        return width;
    }
    int get_height() {
        return height;
    }
    // 현재 인스턴스 사각형과 r이 겹쳐 있다면 true, 그렇지 않으면 false
    bool intersect(Rectangle r) {
        if ((corner.x > r.corner.x + r.width) || (corner.x + width < r.corner.x) || (corner.y >
r.corner.y + r.height) || (corner.y + height < r.corner.y)) {
            return false;
        }
```

```
        else {
            return true;
        }
    }
    // 인스턴스 사각형과 사각형 r이 겹치지 않는 조건은 r의 코너의 x/y 좌표가 인스턴스 사각형의
코너의 x/y 좌표 + 너비/높이보다 클 경우나 인스턴스 사각형의 코너의 x/y 좌표가 r의 코너의 x/y
좌표 + 너비/높이보다 클 경우이다.
    // 대각선의 길이(int 형)를 반환
    int diagonal() {
        // 코드 작성
        return static_cast<int>(pow((pow(width, 2) + pow(height, 2)), 0.5));
    }
    //pow 함수를 통한 ((너비)^2+(높이)^2)^(0.5)의 결과값을 정수형으로 캐스팅하여 반환
    // 사각형의 중심점의 좌표를 IntPoint 형으로 반환
    IntPoint center() {
        // 코드 작성
        return IntPoint(corner.x + (width / 2), corner.y + (height / 2));
    }
    //인수를 코너의 x/y 좌표 + 너비/높이의 절반으로 하여 생성자 사용.
    // 현재 인스턴스 사각형의 내부(경계포함)에 pt가 있으면 true,
    // 그렇지 않으면 false
    bool is_inside(IntPoint pt) {
        // 코드 작성
        if ((corner.x <= pt.x) && (corner.x + width >= pt.x) && (corner.y <= pt.y) && (corner.y
+ height >= pt.y)) {
            return true;
        }
        else {
            return false;
        }
    }
    //pt의 x/y좌표가 인스턴스 사각형의 코너의 x/y좌표와 그 좌표 + 너비/높이 범위 내에 존재하면
true.
};
int main() {
    IntPoint pt1(0, 0);
    IntPoint pt2(2, 2);
    Rectangle rect1(pt1, 4, 3);
    Rectangle rect2(pt2, 5, 6);
    //각종 인스턴스 선언.
    if (rect1.intersect(rect2)) {
        std::cout << "True" << std::endl;
    }
    else {
        std::cout << "False" << std::endl;
    }
    //rec1과 rec2가 교차하는지 확인. 결과값 True.
    if (rect1.is_inside(rect1.center())) {
        std::cout << "True" << std::endl;
    }
    else {
        std::cout << "False" << std::endl;
    }
    //rec1의 중심이 rect1 안에 있는지 확인. 결과값 True.
    std::cout << rect1.diagonal() << std::endl;
```

```
    //rec1의 대각선의 길이 확인.
```

3. }아래의 main 함수가 주석과 같이 동작하도록 Rational 클래스와 필요한 함수들을 정의하라.

```
#include <iostream>
#include <vector>
#include <string>
int main () {
    Rational r1, r2(5), r3(2, 8), r4;
    Print(r1); // prints 0/1
    Print(r2); // prints 5/1
    Print(r3); // prints 1/4

    r4 = Mul(r2, r3); // r4 = r2*r3
    Print(r4); // prints 5/4
    r4 = r2.Add(r3); // r4 = r2+r3
    Print(r4); // prints 21/4

    if(r4.Equal(Rational{42, 8})) std::cout << "Equal" << std::endl;

    std::vector<Rational> v1;
    v1.push_back({1}); v1.push_back({3, 7});
    Print(v1); // prints 1/1, 3/7

    std::string s1 = "C++ programming", s2;
    s2 = NewString(s1); // s2: "***C++ programming***"
    std::cout << s2 << std::endl; // prints ***C++ programming***
}
```

코드 및 설명:

```
#include <iostream>
#include <vector>
#include <string>

class Rational {
    int num, den;
public:
    Rational(int num = 0, int den = 1) : num(num), den(den) {
        int a = std::min(num,den), b = std::max(num, den);
        while (a != 0) {
            int temp = b % a;
            b = a;
            a = temp;
```

```
        }
        num /= a;
        den /= a;
    };
    //약분 과정을 더하여 생성자 사용.
    int getNum() {
        return num;
    }
    int getDen() {
        return den;
    }
    //private 변수를 사용하기 위한 메서드.
    Rational Add(Rational a) {
        return Rational(getNum() + a.getNum(), getDen() + a.getDen());
    }
    //인스턴스 a를 파라미터로 하고, 현재 인스턴스의 num과 den에 a의 num과 den을 각각 더한 두
값을 인수로 한 인스턴스를 반환.
    bool Equal(Rational a) {
        if ((getNum() == a.getNum()) && (getDen() == a.getDen())) {
            return true;
        }
        else {
            return false;
        }
    }
    //인스턴스 a를 파라미터로 하고, 현재 인스턴스와 a의 num과 den이 각각 같으면 참, 아니면
거짓.
};
void Print(Rational r) {
    std::cout << r.getNum() << '/' << r.getDen() << std::endl;
}
//인스턴스 r을 파라미터로 하여 인스턴스의 num과 den을 출력하는 print 함수.
void Print(std::vector<Rational> v) {
    for (int i = 0; i < v.size(); i++) {
        std::cout << v[i].getNum() << '/' << v[i].getDen() << ", ";
    }
    std::cout << "₩n";
}
//인스턴스를 엘리먼트로 하는 벡터 v를 파라미터로 하여 각각의 num과 den을 모두 출력하는 print
함수. (오버로딩)
Rational Mul(Rational a, Rational b) {
    return Rational(a.getNum() * b.getNum(), a.getDen() * b.getDen());
}
//인스턴스 a와 b를 파라미터로 하고, 인스턴스의 num과 den의 각각의 곱을 다시 인수로 한
인스턴스를 반환.
std::string NewString(std::string a) {
    return "***" + a + "***";
}
//문자열 a를 파라미터로 하고, a 앞뒤에 ***을 붙인 문자열을 반환
int main() {
    Rational r1, r2(5), r3(2, 8), r4;
    Print(r1); // prints 0/1
    Print(r2); // prints 5/1
    Print(r3); // prints 1/4
```

```
    r4 = Mul(r2, r3); // r4 = r2*r3
    Print(r4); // prints 5/4
    r4 = r2.Add(r3); // r4 = r2+r3
    Print(r4); // prints 21/4

    if (r4.Equal(Rational{ 42, 8 })) std::cout << "Equal" << std::endl;

    std::vector<Rational> v1;
    v1.push_back({ 1 }); v1.push_back({ 3, 7 });
    Print(v1); // prints 1/1, 3/7

    std::string s1 = "C++ programming", s2;
    s2 = NewString(s1); // s2: "***C++ programming***"
    std::cout << s2 << std::endl; // prints ***C++ programming***
}
```