DAY2
과제 2번 보고서

[매력적인 요약문으로 독자의 시선을 끌어 보세요. 일반적으로 요약은 문서의 내용을 간략하게 정리한 것입니다. 내용을 추가하려면 여기를 클릭하여 입력하면 됩니다.]

2023741024
로봇학부 박건후

목차:

(1) 헤더 파일

```
#include <iostream>
#include <cstdlib>
#include <ctime>
namespace hw2{
double sqrt_newton(double n);

typedef struct point{
    int x;
    int y;
}P;
class cal_points{

    P* points;
    int size;

    public:
    void put_points(P* points, int n);
    void calculate();
    cal_points();

};
}
```

cstdlib 를 인클루드한 이유는 배웠던 c 언어의 헤더 파일을 받아와서 rand()함수를 사용하여 랜덤으로 사용자가 정해준 범위 내의 좌표를 만들기 위해서입니다. 그리고 srand()를 사용해야하므로 c 언어의 time 헤더 파일을 인클루드했습니다.

과제 1 번 헤더 파일에서 hw1 네임스페이스로 감쌌듯, 이번에는 hw2 네임스페이스로 만든 클래스, 함수를 감쌌습니다. sqrt 함수를 사용할 수 없었기에 제곱근을 근사하여 구하는 함수인 sqrt_newton 함수를 만들어 사용하였습니다. 그리고 문제의 요구사항대로 점을 구조체로 구현하기 위해 point 구조체를 만들었습니다. 문제에서 최소값과 최대값을 구하고 그것에 해당하는 각각의 두 점을 구하는 클래스를 만들어야 했기에 이 기능들을 구현한 클래스를 만들었습니다. 랜덤으로 생성된 점들의 배열을 put_points 에서 받고 거리를 계산하고 최소 거리, 최대 거리를 구하며 해당 점을 저장 및 출력하는 기능을 구현한 calculate 메서드를 만들었습니다. 이렇게 파트를 나눠 분담을 하였습니다.

(2) 구현 파일

```cpp
#include "hw2.hpp"
using namespace hw2;
namespace hw2{
double sqrt_newton(double n) {      //sqrt함수를 못 사용하기에 이를 위한
    double y = n /2 ;      // 초기 추정값
    double difference = 0.00001;  // 허용 오차

    while (y * y - n > difference || n - y * y > difference) {
        y = (y + n / y) / 2.0;
    }

    return y;
}
}
```

점과 점의 거리를 구하는 공식에서 루트를 씌워 리턴하는 함수입니다. 하지만 sqrt()를 사용할 수 없어서 완벽한 루트는 아니지만 루트에 대한 근사값을 구할 수 있는 함수를 만들어야 했습니다. 그래서 sqrt_newton이라는 함수를 만들어 루트 근사값을 구하고자 했습니다. 루트 근사값을 구하는 방법을 검색하여 뉴턴이 발명한 근사값을 구하는 방식이 있다는 것을 알게 되었고 해당 페이지에 나온 공식을 활용하여 루트 근사값을 구해야겠다는 생각을 했습니다. 해당 공식은 공식을 반복하면 반복할수록 루트값에 가까워지는 공식입니다. 그래서 이를 while문으로 감싸서 사용하고자 하였고 이를 무한반복을 할 수는 없으니 어느 정도 리미트를 걸어놔야겠다는 생각을 했습니다. 그래서 y*y=n이 되도록 하는 y변수를 설정했습니다. y는 루트 n이 이상적인 목표인 것입니다. 우선 y를 n의 근처로 설정해두고 반복문을 돌려 n의 루트값에 근사를 시키다가 실제 y*y과 n의 차이가 매우 미세히질 때 반복문을 나와서 그 때의 y값을 반환하게 했습니다. 페이지에 대한 링크나 정보는 아래에 놓아놨습니다.

수학 & 코딩 – 뉴턴의 방법으로 근삿값 구하기 - SHA Computing

이 함수의 구현부를 hw2네임스페이스에 넣어서 같은 공간에 선언, 정의를 하여 함수의 정의를 못 찾는 일이 발생되지 않도록 했습니다.

```
void cal_points::put_points(P* points, int n){
        this->points = points;
        this->size=n;
}
```

이제 클래스 메서드의 구현부입니다. put_points에서는 무작위로 만들어진 점들을 저장한 배열을 포인터로 받는 것과 그 배열의 사이즈를 받는 코드를 썼습니다. 결국 초기화를 하는 메서드인 것입니다.

```
void cal_points::calculate(){
        double max=0.0;
        double dx0 = points[0].x - points[1].x;
        double dy0 = points[0].y - points[1].y;
        double min = sqrt_newton(dx0*dx0 + dy0*dy0);
        double result=0.0;
        P max_points[2] = {points[0], points[1]};
        P min_points[2] = {points[0], points[1]};
```

calculate메서드에서는 과제 1번처럼 max, min변수를 만들고 이번에는 해당 점들도 저장해야 하기에 max인 값에 해당되는 두 점을 저장하는 max_points배열, min인 값에 해당되는 두 점을 저장하는 min_points 배열을 만들고 초기화하였습니다. 그리고 max, min 값을 초기화할 때, 거리값은 항상 0이상이므로 max를 0.0으로 초기화하면 되는 반면, min값을 초기화할 때는, 상수값을 초기화하게 되면, 모든 점들의 거리가 그 상수값보다 크다면 min값이 변경되지 않으므로 임의의 점인 0, 1번째 점을 가지고 거리를 계산하여 초기화하였습니다.

```
    for(int i=0; i< size; i++ ){
        for(int j=i+1; j<size; j++){
            result = sqrt_newton((points[i].x - points[j].x)*(points[i].x - points[j].x)
            if(max< result){
                max= result;
                max_points[0] = points[i];
                max_points[1] = points[j];
            };
            if(min>result){
                min = result;
                min_points[0] = points[i];
                min_points[1] = points[j];
            }

        }
    }
```

이제 초기화를 모두 끝내고 모든 점들의 거리를 계산하고, max, min 값이 최신화될 때 해당되는 점들의 정보를 max_points, min_points 배열에 저장하도록 했습니다. 그리고 점들끼리 거리를 계산할 때는 하나의 점을 잡고, 그 뒤의 요소들과 모두 거리를 재는 방식을 사용했습니다. 이것을 각각의 요소마다 수행하는 것입니다. 그렇게 되면 중복되는 계산을 피할 수 있으며 빠짐없이 각 점들 간의 거리를 잴 수 있기 때문에 사용하였습니다.

```
        }
        std::cout<<"--------Result--------"<<std::endl<<"MinDist= "<<min<<std::endl<<"Pair of Min Coor.(x,y)
        std::cout<<"MaxDist= "<<max<<std::endl<<"Pair of Max Coor.(x,y): "<<"P1("<<max_points[0].x<<","<<max
}
cal_points::cal_points():points(NULL), size(0){};
```

그 후에는 최대, 최소값을 출력하고, 해당 점들을 문제 예시에 맞게 출력하는 부분입니다. 그 아래에는 기본 생성자를 정의한 것입니다. 이번에도 처음 객체가 생성되었을 때 멤버변수에 쓰레기값이 들어가지 않도록 하였습니다.

(3) main 파일

```
#include "hw2.hpp"
using namespace hw2;

int main(){
    srand((unsigned)time(NULL));
    int num=0;
    int max_coor = 0;
    int min_coor = 0;
    int nx=0;
    int ny=0;


    std::cout<<"please define the number of points: ";
    std::cin>> num;
    while(!std::cin || num<2){
            std::cout<<"잘못된 입력입니다. 다시 입력하세요"<<std::endl;
            std::cin.clear(); //오류 상태 초기화
            std::cin.ignore(1000, '\n'); //버퍼에 들어간 문자 무시
            std::cout<<"please define the number of points: ";
            std::cin>> num;
    }
```

num 이라는 변수로 몇 개의 점을 생성할 것인지 입력하는 부분입니다. 이 때 생성되어야 할 점의 개수는 2 개 미만이면 안 됩니다. 이에 대한 것과 문자가 입력되었을 때나 잘못 입력되었을 때에 대한 예외처리를 진행하였습니다. 이번에는 while 문으로 진행하여 cin 을 정상상태로 만들고 다시 입력을 받는 것을 반복시켰습니다.

```
 std::cout<<"please define the number of coor, value: ";
 std::cin >> min_coor;
 while(!std::cin){
        std::cout<<"잘못된 입력입니다. 다시 입력하세요"<<std::endl;
        std::cin.clear(); //오류 상태 초기화
        std::cin.ignore(1000, '\n'); //버퍼에 들어간 문자 무시
        std::cout<<"please define the number of coor, value: ";
        std::cin>> min_coor;
 }
```

이번에는 min_coor 로 점의 최소값을 정하는 부분입니다. 이에 대해서는 특별히 값에 대한 예외처리가 없기에 !std::cin 으로 예외처리를 하였습니다.

```
std::cout<<"please define the number of coor, value: ";
std::cin >> max_coor;
while(!std::cin || max_coor<=min_coor){
        std::cout<<"잘못된 입력입니다. 다시 입력하세요"<<std::endl;
        std::cin.clear(); //오류 상태 초기화
        std::cin.ignore(1000, '\n'); //버퍼에 들어간 문자 무시
        std::cout<<"please define the number of coor, value: ";
        std::cin>> max_coor;
}
```

이번에는 max_coor 에 대해 좌표의 최댓값을 입력받아 설정하는 부분입니다. 이 때 min_coor 보다 값이 커야 하므로 그렇지 않은 경우에 대한 예외처리를 하였습니다. 그래서 while 조건식에 max_coor<=min_coor 가 들어가게 됩니다.

```
std::cout<< "Generate Random points"<<std::endl;
P* points = new P[num];
for(int i=0; i< num; i++){
    nx = rand()%(max_coor-min_coor) + min_coor;
    ny = rand()% (max_coor - min_coor)+min_coor;
    std::cout<<"Point "<<i<<". nX= "<<nx;
    std::cout<<", nY= "<< ny<<std::endl;
    points[i] = P{nx, ny};
}
```

이제 입력받은 num 과 범위로 무작위로 점을 생성할 차례입니다. 생성할 점 만큼 동적할당을 points 라는 포인터에 진행하고 rand()로 해당 범위 안의 값이 산출되도록 나머지 연산 등의 추가적인 연산을 진행하였습니다. 그리고 난수로 생성된 값으로 구조체 임시 객체를 만들고 이를 points 가 가리키는 동적할당 배열에 저장하였습니다.

```
    cal_points cal;
    cal.put_points(points, num);
    cal.calculate();
    std::cout<<"********completed*********"<<std::endl;
    delete points;

    return 0;
}
```

이제 문제에서 원하는 결과를 내주는 클래스에 대한 객체를 생성하고 점에 대한 정보를 넣은 후 계산하여 출력하게 하였습니다.

(4) 실행 사진

```
geonhu@PGH:~$ cd intern_ws/cpp_test/build
geonhu@PGH:~/intern_ws/cpp_test/build$ cmake ..
-- Configuring done
-- Generating done
-- Build files have been written to: /home/geonhu/intern_ws/cpp_test/build
geonhu@PGH:~/intern_ws/cpp_test/build$ make
Consolidate compiler generated dependencies of target test
[ 33%] Linking CXX executable test
[100%] Built target test
geonhu@PGH:~/intern_ws/cpp_test/build$ ./test
please define the number of points: -10
잘못된 입력입니다. 다시 입력하세요
please define the number of points: c
잘못된 입력입니다. 다시 입력하세요
please define the number of points: ef
잘못된 입력입니다. 다시 입력하세요
please define the number of points: 5
please define the number of coor, value: er
잘못된 입력입니다. 다시 입력하세요
please define the number of coor, value: -20
please define the number of coor, value: -30
잘못된 입력입니다. 다시 입력하세요
please define the number of coor, value: 50
Generate Random points
Point 0. nX= 7, nY= 24
Point 1. nX= 31, nY= 6
Point 2. nX= 3, nY= 29
Point 3. nX= -4, nY= 13
Point 4. nX= 39, nY= -17
--------Result--------
MinDist= 6.40312
Pair of Min Coor.(x,y): P1(7,24) & P2(3,29)

MaxDist= 58.4123
Pair of Max Coor.(x,y): P1(3,29) & P2(39,-17)

********completed*********
```