Problem 1 : Numeric diamonds

프로그램의 전반적인 알고리즘

(1) size 를 입력 받는다.

(2) 1 부터 $(size)^2$ 까지의 숫자를 정사각형의 형태로 출력한다.

(3) 4 가지의 기본명령어 중 하나를 입력한다.

(4) 행당 명령을 실행한다.

(5) (3)-(4)번을 반복한다.

요구사항

1. 입력:

A. size 는 자연수로 입력 받는다.

```
1:  int flag = 0;
2:  while (flag != 1) {
3:          if (flag != 1) {
4:                  printf("Enter size: ");
5:          }
6:          flag = scanf("%d",&size);
7:          FLUSH;
8:  }
```

입력된 값이 실수가 나올 때까지 입력을 받는다. 이 때 실수형의 정수형 자리만 size 의 크기로 지정된다. FLUSH;를 사용함으로써 buffer 에 남아있는 문자를 없앤다.

B. 4 가지 기본명령어

```
1:  int command='c', rot = 0;
2:  while (1) {
3:          if (command == 'c') {
4:                  size = get_size();
5:                  print_square(size);
6:                  rot = 4;
7:          }
8:          command = get_command();
9:          if (command == 'q') {
10:                 break;
```

① c

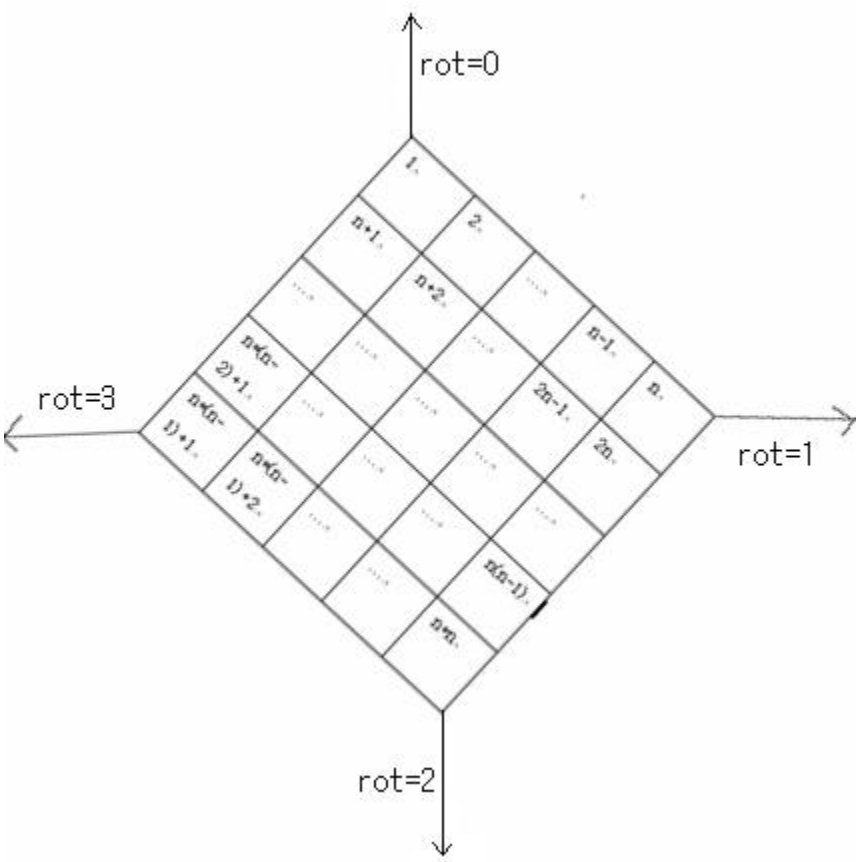
'c'가 입력되었을 경우, 새로운 size를 입력 받고, 정사각형의 모양으로(size*size)출력시킨다. rot을 0으로 초기화한다.

② q

'q'가 입력되었을 경우, while 문을 빠져나감으로써, 프로그램을 끝낸다.

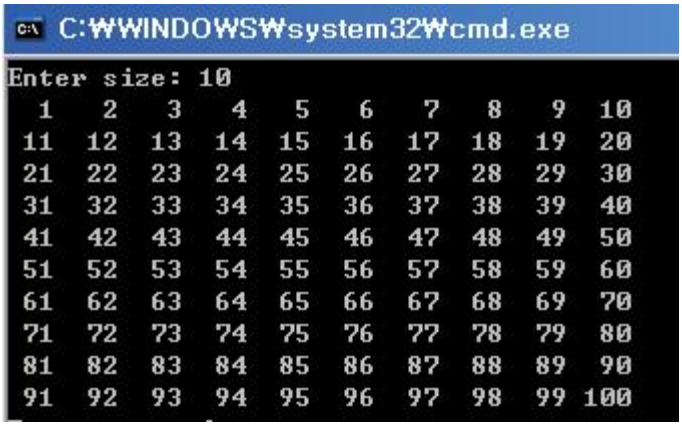
11:	}	③ r 다이아몬드 모양을 오른쪽으로 90도 회전시킨다. (rot을 3증가시킴)
12:	else if (command == 'r') {	
13:	rot =(rot + 4 – 1)% 4;	
14:	print_dia(size, rot);	
15:	}	
16:	else if (command == 'l') {	④ l 다이아몬드 모양을 왼쪽으로 90도 회전시킨다. (rot 을 1 증가시킴)
17:	rot = rot + 1;	
18:	rot = rot % 4;	
19:	print_dia(size, rot);	
20:	}	
21:	}	

- rot:

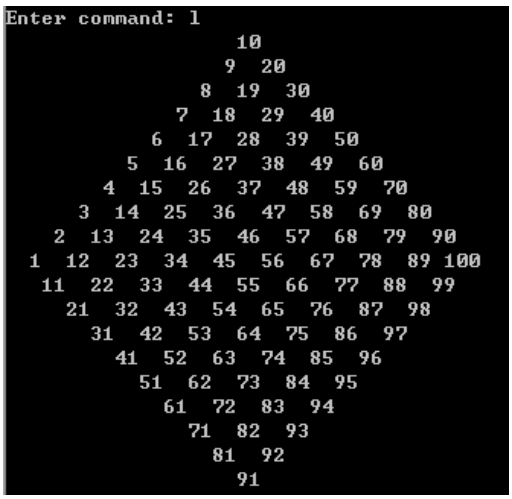


2. 출력

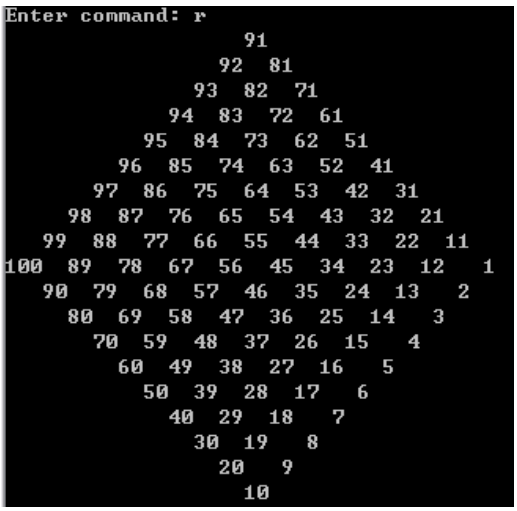
Step 1: size 입력 (ex) n=10)



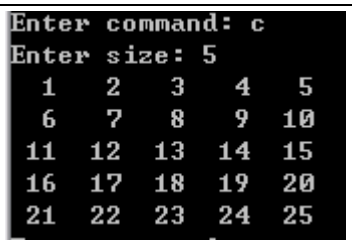
Step 2: 사용자 입력 (l,r,c,q) 중 하나를 받고 각각의 명령을 수행한다.



Case 1. 사용자 입력: l



case 2. 사용자 입력: r



Case 3. 사용자 입력: c

Case 4. 사용자 입력: q

3. 입력이 l, r, q, c와 같이 알파벳인 경우에는 대소문자를 구분하지 않습니다.

char tolower(char var) {
if (var >= 'A' && var <='Z')
{
return var + 32;
}
return var; }
	command=tolower(command);
	입력받은 문자를 tolower()함수를 호출하여 대문자를 소문자로 바꿈으로써 대소문자의 구분을 없앤다.

4. 아직 수업시간에 다루지 않은 배열은 사용하지 않았습니다.

5. 입력과 출력에 사용되는 함수 (e.g. scanf, printf, …) 이외의 library function은 사용하지 않았습니다.

6. 입력이 l 과 r이 주어졌을 때, 회전하는 함수를 호출하도록 합니다.

l이 주어졌을 때, rot을 3증가시키고, 4로 나눈 나머지를 rot으로 지정하고, r이 주어졌을 때, 1을 증가시키고, 4로 나눈 나머지를 rot에 저장한다. 출력함수는 아래와 같으며, size와 rot에만 의존하는 함수로, l,r의 경우를 모두 처리하도록 한다.

void print_dia(int size, int rot) { int a,b,c;

if (rot == 0) {
for (a=1 ; a <= size ; a++) {
for (c=1; c<= size-a; c++)
printf(" ");
for (b = 1; b <= a; b ++) {
printf("%3d", ①(a - b)*size + b);
printf(" "); }
printf("₩n"); }
for (a=1 ; a<=size-1 ; a++) {
for (c=1; c<= a; c++)
printf(" ");
for (b = 1; b <= size - a; b ++){
printf("%3d",②(size - b)*size+(b + a));
printf(" "); }
printf("₩n"); }
}
	If(rot==1)
	➔ ①: b*size + (b-a)
	➔ ②: (a + b - 1)*size + b
	If(rot==2)
	➔ ①: (size - a + b)*size - b +1
	➔ ②: (b)*size - (a +b-1)
	If(rot==3)
	➔ ①: (size-b)*size + (a-b+1))
	➔ ②: (size - (a+b-1))*size - (b-1)

->program 구현: a는 줄 번호, b는 칸 번호

위의 rot 번호를 매긴 그림에서 각각 줄 번호와 칸 번호에 대해 일반항을 찾는다. 그리고 for문을 이용하여 그대로 출력을 하였다. 이 때 빈칸을 출력해 줘야 하는데, 빈칸은 줄 번호와 관련이 있으므로 줄 번호에 비례하여 출력해 주었다.

***추가기능!!!**

r을 입력 받으면 오른쪽으로 회전, l을 입력 받으면 왼쪽으로 회전하게 되는데, 항상 처음상태를 기준으로 회전하지 않고, 회전한 상태에서 또다시 회전을 하는 기능을 추가했다.

* 토론

- 출력 시 정확한 다이아몬드 모양을 얻기 위해 print()함수 사용시 %d 대신에 %3d를 써서 출력을 하였다.
- 추가기능을 사용하여 r, l을 여러 번 눌렀을 때에 전 단계와 이어지도록 변수 rot을 사용하여 구현을 해보았다.

* 결론

- 반복 문에 대해서 연습할 수 있는 좋은 기회가 되었고, 반복 문을 사용하는 방법에 대해서 익숙해 질 수 있었다.

* 개선방향

- 오른쪽, 왼쪽으로 회전하는 것 이외에 다이아몬드 모양의 중심선을 기점으로 위 아래를 뒤바뀌게 하는 알고리즘도 구현해 볼 수 있을 것이다. 이는 r을 두 번 연속 하는 것 혹은 l을 두 번 연속 하는 모양과 다르게 나옴으로써 다른 배열의 다이아몬드 모양을 생성할 수 있다.

Problem 2 - Inside? Outside?

● 프로그램의 전반적인 알고리즘

(1) 다각형 점의 정보를 담고 있는 파일을 연다. (열리지 않으면 프로그램 종료)

(2) 테스트할 점의 정보를 담고 있는 파일을 연다. (열리지 않으면 프로그램 종료)

(3) 결과출력 파일을 연다. (쓸 수 없으면 프로그램 종료)

(4) 테스트할 점과 다각형의 두 점을 이은 선분과 위치를 비교한다. (singed area 이용. sa < 0 이면 오른쪽)

(5) 다각형의 모든 선분을 이용해 (4)를 반복한다. 한 선분이라도 위치가 왼쪽에 있다면 outside 판정 이 때 outside 판정이 나면 결과파일에 출력한다.

(6) (5)를 모두 반복하여 항상 오른쪽에 있다면 inside 판정 이 때 inside 판정이 나면 결과파일에 출력한다.

(7) 다음 테스트할 점에 대해 (4)의 과정을 반복한다.

(8) 열려진 파일을 닫고 프로그램을 종료한다.

● **요구사항**

1. 입력:

```
1:  fscanf(polygon_file,"%d",&polygon_num);
2:  flag = fscanf(polygon_file,"%lf%lf",&x1, &y1);
3:  if (flag == EOF) {
4:  return 2;
5:  }
6:  for (i=1 ; i <= polygon_num ; i++) {
7:  flag = fscanf(polygon_file,"%lf%lf",&x2, &y2);
8:  if (flag == EOF) {
9:  return 2;
10: }
```

A. File1 : 다각형의 꼭지점 좌표가 들어있는 파일 (별첨 polygon.dat)

polygon.dat 의 파일 포인터는 polygon_file 이다. 여기서 맨 처음은 다각형 점의 개수가 저장되어있다. 그래서 1 번줄과 같이 처리를 하였다. 그리고 다음 line 는 점의 좌표가 x, y 로 저장되어 있으므로 fscanf 오 %lf%lf 형식으로 읽어 들였다.

```
1:  flag = fscanf(test_file, "%d%lf%lf",&test_point_index, &x, &y);
```

```
2:  while (flag != EOF) {
3:  rewind(polygon_file);
4:  in_or_out = in_out_check(polygon_file, x, y);
5:  if (in_or_out == 2) {
6:  printf("!! 다각형의꼭지점좌표가들어있는파일형식이올바르지않습니다!!₩n");
7:  return 0;
8:  }
9:  point_write(output_file, test_point_index, in_or_out);
10: flag = fscanf(test_file, "%d%lf%lf",&test_point_index, &x, &y);
11: }
```

B. File2 : 테스트할 점들의 좌표가 들어있는 파일 (별첨 indata.dat)

indata.dat 의 파일포인터는 test_file 이다. 가장 처음은 점의 번호이고, x 좌표, y 좌표이다. fscanf 로 %d%lf%lf 로 읽어 들였다.

2. 출력

```
1:  if (in_or_out == 1) {
2:  fprintf(output_file, "%d inside₩n", test_point_index);
3:  }
4:  else {
5:  fprintf(output_file, "%d outside₩n", test_point_index);
6:  }
```

A. Output file : 테스트 결과를 출력한 파일

20071014.dat 파일로 결과를 내놓는다. 20071014.dat 의 파일포인터는 output_file 이다. in_or_out 은 테스트한 점이 다각형의 내부에 있는지 외부에 있는지 결과를 가진다. 1 일 때는 내부를 뜻한다. 그러므로 경우로 나누어서 fprintf 함수를 이용해 정해진 형식으로 출력하였다.

3. 프로그램 구현

여러 함수를 정의하여 사용하였다.

FILE* fopen_polygon(); // polygon.dat 파일을 여는 함수

FILE* fopen_test(); // indata.dat 파일을 여는 함수

FILE* fopen_output(); // 20071014.dat 결과파일을 여는 함수

```
int in_out_check (FILE* polygon_file, double x, double y); // 점이 다각형 내부인지 외부인지 판단
void point_write(FILE* output_file, int test_point_index, int in_or_out); // 점의 결과를 출력함
```

```
1:  flag = fscanf(test_file, "%d%lf%lf",&test_point_index, &x, &y);
2:  while (flag != EOF) {
3:  rewind(polygon_file);
4:  in_or_out = in_out_check(polygon_file, x, y);
5:  if (in_or_out == 2) {
6:  printf("!! 다각형의꼭지점좌표가들어있는파일형식이올바르지않습니다!!₩n");
7:  return 0;
8:  }
9:  point_write(output_file, test_point_index, in_or_out);
10: flag = fscanf(test_file, "%d%lf%lf",&test_point_index, &x, &y);
11: }
```

프로그램에서 배열을 쓰지 않았다. 그러므로 한 개의 점을 테스트 할 때 마다 다각형 점의 좌표를 새로 읽어 들여야 했다.

테스트할 파일을 fscanf 로 읽어들였다. 그리고 함수의 리턴값을 EOF 와 비교해서 파일이 끝났는지 고려하였다. 한 개 점을 테스트할 때 마다 다각형 점 리스트를 새로 읽어 들여야 하므로 rewind() 함수를 이용해 파일마커를 가장 처음으로 되돌려 주었다. in_out_check() 함수에서 그 점이 다각형 내부인지 외부인지 테스트를 한다. 만약 이 함수의 리턴값이 2 가 되면 각각 파일형식이 올바르지 않다고 판단하여 프로그램이 종료된다.

한 개의 점을 테스트할 때 마다 출력파일에 point_write 함수를 이용해 기록한다.

- 한 개 점을 테스트하는 방법

```
1:  flag = fscanf(polygon_file,"%lf%lf",&x1, &y1);
2:  if (flag == EOF) {
3:  return 2;
4:  }
```

```
5:  for (i=1 ; i <= polygon_num ; i++) {
6:  flag = fscanf(polygon_file,"%lf%lf",&x2, &y2);
7:  if (flag == EOF) {
8:  return 2;
9:  }
10: if ((x2-x1)*(y-y1)-(x-x1)*(y2-y1) >= 0) {
11: return 0;
12: }
13: x1 = x2;
14: y1 = y2;
15: }
```

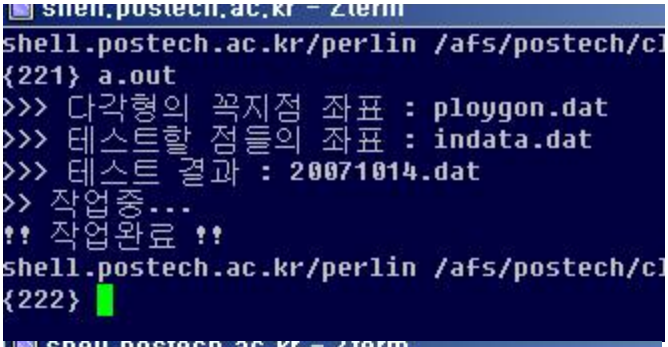
x 와 y 변수는 테스트할 점의 좌표이다. 그리고 polygon_num 는 다각형 점의 좌표 개수를 담고 있다. 그리고 한 개의 다각형 점의 좌표를 읽고 x1 과 y1 에 저장한다. 그다음 for 문을 이용해 다각형 점의 좌표를 계속 읽어 들인다. x2 와 y2 에 저장을 하고 (x2-x1)*(y-y1)-(x-x1)*(y2-y1) 값의 부호를 체크한다. 0 이라면 점이 선분위에 있다는 것이고, 0 보다 크면 왼쪽에 있다는 뜻이다. 그러므로 체크를 하는 도중 이 값이 0 이상이 되면 outside 로 판정이 나므로 0 을 리턴해 준다. 만약 for 문을 모두 거친 후에는 모든 선분이 0 보다 작은 결과를 가진다는 뜻이므로 1 을 리턴해 준다.

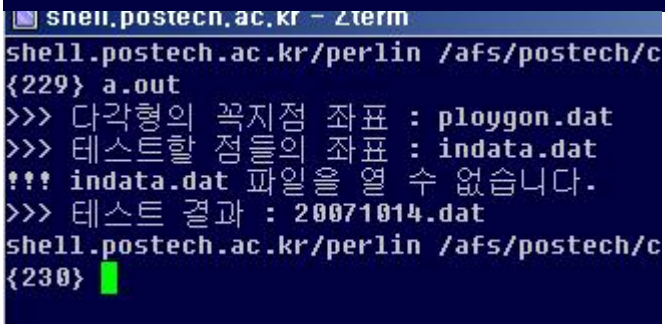
- 파일을 읽고 쓸 때

```
1:  polygon_file = fopen("polygon.dat","r");
2:  if (polygon_file == NULL) {
3:  printf("!!! polygon.dat 파일을 열 수 없습니다.₩n");
4:  }
```

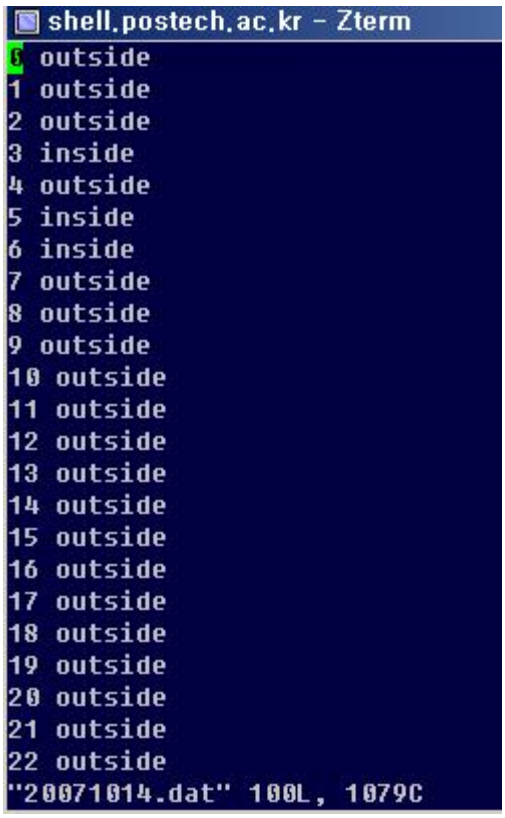
파일을 쓰고 열 때 fopen 함수를 사용하였다. 만약 읽으려는 파일이 없을 경우 fopen 함수의 리턴값은 NULL 이 되므로 NULL 과 체크를 하여 파일을 열 수 없을 때 에러처리를 하였다.

● 실행 모습



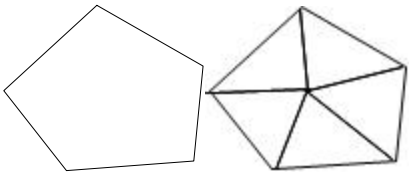


왼쪽의 그림은 모든 파일이 정상적으로 있을 때의 성공적으로 테스트를 한 결과이다. 오른쪽은 indata.dat 파일이 없을 때 프로그램이 종료되는 모습을 나타낸다. 그리고 생성된 20071014.dat 파일을 vi 로 열어보면 다음과 같다.



● 추가기능

문제에서 제시한 inside/outside 판정 방법 이외에 다른 방법을 고안하였다.



왼쪽의 그림과 같이 하나의 볼록 다각형의 각각 좌표를 이용해 넓이(S)를 계산할 수 있다. 그리고 오른쪽의 그림과 어떤 다각형의 내부의 점에서 각각 다각형의 꼭지점을 연결하면 삼각형을 여러 개 만들 수 있다. 이 삼각형의 넓이의 합이 S 와 같으면 내부의 점이 되고, 그렇지 않은 경우에는 외부의 점이 되는 것이다.

점의 좌표가 시계방향 순서대로 (x1, y1) , (x2, y2) .... (xn , yn) 이라고 하면, 이 다각형의 넓이 S는 다음과 같은 공식으로 구할 수 있다.

$$2S = |(x_1y_2 - x_2y_1) + (x_2y_3 - x_3y_2) + \cdots + (x_{n-1}y_n - x_ny_{n-1}) + (x_ny_1 - x_1y_n)|$$

이러한 방법을 이용해 추가기능을 구현 하였다. 또한 polygon.dat 파일의 형식은 가장 처음에 주어진 점이 마지막에도 나오므로 x 와 y 의 index 번호에 주의를 기울이면 다각형 넓이를 구하는 것이 가능해 진다.

```
1:  int polygon_area(FILE* polygon_file) {
2:  int i, flag;
3:  int polygon_num;
4:  int x_k1, y_k1, x_k, y_k;
5:  int sum = 0;
6:  int double_area = 0;
7:
8:  rewind(polygon_file);
9:  fscanf(polygon_file,"%d",&polygon_num);
10:
11: flag = fscanf(polygon_file,"%d%d",&x_k, &y_k);
12: if (flag == EOF) {
13: return 2;
14: }
15: for (i=1 ; i <= polygon_num ; i++) {
16: flag = fscanf(polygon_file,"%d%d", &x_k1, &y_k1);
```

```
17:  if (flag == EOF) {
18:  return 2;
19:  }
20:  sum = sum + (x_k * y_k1 - x_k1 * y_k);
21:  x_k = x_k1;
22:  y_k = y_k1;
23:  }
24:  double_area = abs(sum);
25:  return double_area;
26:  }
```

절대값을 사용하기 위해 math.h 파일을 include 하였고, abs 함수를 사용하였다. 각 점을 하나씩 읽어 들이고, 파일형식이 가장 처음 점의 좌표가 마지막에도 한번 나오기 때문에 구현이 편리하였다. 원래 넓이를 구할 때 2 로 나누어 줘야하지만, 모든 넓이를 2 로 나누지 않고 처리를 하면 int 값으로 바로 처리가 가능하기 때문이다.

그리고 내부 점의 좌표에 대해서 삼각형의 넓이를 구할 때 방법은 간단하므로 따로 함수를 만들었다.

```
int triangle_area (int x1,int y1,int x2,int y2,int x3,int y3) {
int double_area = 0;
double_area = (x1*y2-x2*y1) + (x2*y3-x3*y2) + (x3*y1-x1*y3);
double_area = abs(double_area);
return double_area;
}
```

파라미터 값은 각각 세 점의 좌표이다. 역시 위에서 제시한 공식에 의해 간단히 넓이를 계산 가능하다.

파일에서 좌표를 읽어오는 원리는 처음 기능과 동일하다. 그래서 비교적 쉽게 구현이 가능했다.

2. 토론

● 추가기능과 효율성 비교추가기능 알고리즘과 제시한 방법중 어느것이 효율적일지 고려해 보았다. 두 방법 모두 동일한 수 만큼의 for 반복문을 사용했다. 문제에서 제시한 방법은

비교적 계산이 간단하고 부호에만 신경쓰면 되므로 좀더 효율적일 것이다. 하지만 추가기능에서 사용한 방법은 비교적 덧셈과 곱셈의 계산과정이 많고 그 값을 다시 비교해야 하므로 덜 효율적이라고 생각된다. 그래도 추가기능과 같이 내부/외부 판정하는 방법은 그림으로 그려보면 한눈에 이해가 되므로 어떤 방식을 사용했는지 알기가 편할 것 같다.

3. 결론

파일의 입출력에 대해서 잘 알게 되었다. 그리고 rewind()함수를 잘 사용해서 파일 입력이 용이하다는 것을 알게 되었다. 문제에서 제시한 방법 외에 또다른 방법을 생각해 볼 수 있는 기회가 되어서 좋았다.

4. 개선 방향

현재 주어진 다각형의 점들은 순서가 잘 정렬 되어있고, 오목 다각형이 아니다. 그러므로 내부/외부 판정이 쉬웠다. 하지만 순서가 정렬되어있지 않은 볼록다각형의 경우에는 각각의 점을 방향성을 띄도록 정렬하는 기능이 있어야 할 것 같다.

오목 다각형 일 때는 문제가 매우 복잡해 질 것 같다. 오목다각형을 적절하게 몇 개의 볼록다각형으로 나눌 수 있을 것이고 그 볼록다각형에 대해 내부/외부 판정을 하면 될 것 같다. 하지만 프로그램의 구현이 매우 어려울 것이라고 생각된다.