Interim feedback on triangle assignment

This is just temporary feedback on the automarking of the triangle program, not any extension work.

Your triangle program passes all of the 50 tests.

Your lines program tackles a nice problem. It has some long, straggly and repetitive code, and no autotesting. To answer your question, if you use gradients you have to check for special cases (infinite gradient) before the general case. You could use some other representation such as angle or rational pair which doesn't have special cases (though there are still precision or normalization issues). Or you could manipulate the equation for lines being parallel so that it has no factor underneath, and therefore no special cases, i.e. h1/v1 == h2/v2 becomes h1*v2 == h2*v1. I've added 15% for this.

General notes

My neatest solution to the problem uses a convert function like this:

```
// Convert a string to a number.  If the original string was not in the
// proper format, or if the number is out of range, return 0.
long convert(const char length[]) {
    long n = atol(length);
    char s[21];
    sprintf(s, "%ld", n);
    if (strcmp(s, length) != 0) n = 0;
    if (n < 0 || n > 2147483647) n = 0;
    return n;
}
```

It would have been a bit better to use `INT_MAX` from `limits.h`, but I was trying to keep things simple. It combines validation with conversion to avoid multiple conversions of the same input string. And it uses a bit of trickery to avoid loops or complexity. Without the trickery, I would write:

```
// Convert a string to a number.  If the original string was not in the
// proper format, or if the number is out of range, return 0.
long convert(const char length[]) {
    long n = atol(length);
    for (int i=0; i<strlen(length); i++) {
        if (! isdigit(length[i])) n = 0;
    }
    if (strlen(length) > 1 && length[0] == '0') n = 0;
    if (n > 2147483647) n = 0;
    return n;
}
```

Then my triangle function is:

```
// Classify a triangle, given side lengths as strings:
int triangle(const char sa[], const char sb[], const char sc[]) {
    int type;
    long a = convert(sa), b = convert(sb), c = convert(sc);
    if (a <= 0 || b <= 0 || c <= 0) type = Illegal;
    else if (a + b < c || a + c < b || b + c < a) type = Impossible;
```

```
    else if (a + b == c || a + c == b || b + c == a) type = Flat;
    else if (a == b && b == c) type = Equilateral;
    else if (a == b || a == c || b == c) type = Isosceles;
    else if (a*a+b*b==c*c || a*a+c*c==b*b || b*b+c*c==a*a) type = Right;
    else type = Scalene;
    return type;
}
```

This has one line for each test, and avoids early returns. If the tests were even slightly more complex, it would be worth having a separate function for each triangle type. The order of the tests is critical, making each test simpler, because you can assume that the previous ones have failed. The function does not sort the lengths, because it is barely worth it. If you wanted to do that, the best way would be to write a separate sort function and pass the lengths in an array.

The `long` type is used so that expressions like `x*x + y*y` are guaranteed not to overflow if `x` and `y` are within the range of the `int` type. It is marginal: `int` values are less than 2^{31}, so the value of `x*x + y*y` is less than $2*2^{62} = 2^{63}$ so is just guaranteed to be within the `long` range.

The last few tests check whether any inadequate techniques have been used to avoid overflow. For example, using `double` isn't good enough because it only has 53 bits of precision.

Compiler errors

Some of you submitted a program where the compiler gives warnings. A warning usually indicates a poor design issue or a dangerously bad habit, even if it doesn't prevent the program working. So you need to compile with `-Wall` and fix up the program until there are no warnings. I didn't have time to explain the warnings well in the individual feedback, so ask if you don't understand them or know how to fix them.