Step 1: Function Implementation ([slope.cpp](https://github.com/IntegralWorks/CSCI_4330_Assignment6/blob/main/slope.cpp))

(Note to graders: All code is here: <https://github.com/IntegralWorks/CSCI_4330_Assignment6/tree/main>)

#include <iostream>

#include <regex>

#include <string>

#include <cmath>

#include <limits>

#include <vector>

#include <stdexcept>

#include <cfloat>

*enum* input\_indices

{

    X1\_VALUE = 1,

    Y1\_VALUE,

    X2\_VALUE,

    Y2\_VALUE,

    ROUNDING\_PLACE

};

typedef *enum*

{

    NO\_ERROR,

    DIV\_BY\_ZERO,

    INFINITY\_REACHED,

    INVALID\_COORD\_INPUT,

    INVALID\_ROUNDING\_PLACE\_INPUT

} error\_code\_t;

typedef *struct*

{

    error\_code\_t x1\_value\_check = NO\_ERROR;

    error\_code\_t y1\_value\_check = NO\_ERROR;

    error\_code\_t x2\_value\_check = NO\_ERROR;

    error\_code\_t y2\_value\_check = NO\_ERROR;

} input\_sanitizer\_checks\_t;

typedef *struct*

{

*double* x;

*double* y;

} point\_t;

typedef *struct*

{

    error\_code\_t error\_code = NO\_ERROR;

*double*       value      = 0.0;

} result\_t;

*void* print\_error\_code(error\_code\_t *error\_code*)

{

    switch(*error\_code*)

    {

        case NO\_ERROR:

            std::cout<<"Input is good!\n";

            break;

        case DIV\_BY\_ZERO:

            std::cout<<"error: DIV\_BY\_ZERO\n";

            break;

        case INFINITY\_REACHED:

            std::cout<<"error: INFINITY\_REACHED\n";

            break;

        case INVALID\_COORD\_INPUT:

            std::cout<<"error: INVALID\_COORD\_INPUT\n";

            break;

        case INVALID\_ROUNDING\_PLACE\_INPUT:

            std::cout<<"error: INVALID\_ROUNDING\_PLACE\_INPUT\n";

            break;

        default:

            std::cout<<"error: ERROR CODE NOT FOUND!\n";

            break;

    }

}

error\_code\_t input\_sanitizer(const std::vector<std::string> &*value\_strs*, const std::regex& *float\_pattern*)

{

    for (*int* i = 0; i < 4; ++i)

    {

        if (std::regex\_search(*value\_strs*[i], *float\_pattern*))

        {

            if ( (abs(std::stod(*value\_strs*[i])) >= 1.79769e+308) || (abs(std::stod(*value\_strs*[i])) <= 2.22507e-308) )

            {

                std::cout<<std::stod(*value\_strs*[i])<<'\n';

                return INFINITY\_REACHED;

            }

        }

        else

        {

            return INVALID\_COORD\_INPUT;

        }

    }

    return NO\_ERROR;

}

*void* set\_point(point\_t \**pt*, std::string *x\_value\_str*, std::string *y\_value\_str*)

{

*pt*->x = std::stod(*x\_value\_str*);

*pt*->y = std::stod(*y\_value\_str*);

}

*double* round\_to\_specific\_place(*double* *value*, *int* *rounding\_place*) {

*double* f = std::pow(10.0, *rounding\_place*);

    return std::round(*value* \* f) / f;

}

result\_t calculateSlope(point\_t \**pt1*, point\_t \**pt2*, *int* *rounding\_place*)

{

    result\_t result;

*double*   numerator, denominator;

    // //dev-test

    // std::cout << "\n[dev-test] raw point values\n------------\n";

    // std::cout << '|' << pt1->x << '\n';

    // std::cout << '|' << pt1->y << '\n';

    // std::cout << '|' << pt2->x << '\n';

    // std::cout << '|' << pt2->y << '\n';

    // // std::cout << "\n------------\n";

    numerator   = *pt2*->y - *pt1*->y;

    denominator = *pt2*->x - *pt1*->x;

    if ((*int*)denominator == 0)

    {

        result.error\_code = DIV\_BY\_ZERO;

        return result;

    }

    result.error\_code = NO\_ERROR;

    result.value      = round\_to\_specific\_place(numerator/denominator, *rounding\_place*);

    return result;

}

*int* main(*int* *argc*, *char* \**argv*[])

{

    //input format: <x1> <y1> <x2> <y2> <rounding\_place>

    std::regex float\_pattern(R"([0-9]+\.[0-9]+(?:e[+-]?[0-9]{1,308})?)");

    std::vector<std::string> value\_strs = {*argv*[X1\_VALUE], *argv*[Y1\_VALUE], *argv*[X2\_VALUE], *argv*[Y2\_VALUE]};

    // //dev-test

    // std::cout << "\n[dev-test] raw string values\n------------\n";

    // for (int i = 0; i < 4; ++i)

    // {

    //  std::cout << '|' << value\_strs[i]<<'\n';

    // }

    // // std::cout << "\n------------\n";

    error\_code\_t value\_input\_check = input\_sanitizer(value\_strs, float\_pattern);

    if (value\_input\_check != NO\_ERROR)

    {

        print\_error\_code(value\_input\_check);

        return 1;

    }

*int* rounding\_place;

    try

    {

        rounding\_place = std::stoi(*argv*[ROUNDING\_PLACE]);

    }

    catch (...)

    {

        print\_error\_code(INVALID\_ROUNDING\_PLACE\_INPUT);

    }

    if (rounding\_place < 0 || rounding\_place >= 16)

    {

        std::cout<<"[ERROR] Rounding place range is strictly [0,15] \n";

        print\_error\_code(INVALID\_ROUNDING\_PLACE\_INPUT);

        return 1;

    }

    std::cout<<rounding\_place<<'\n';

    point\_t pt1, pt2;

    set\_point(&pt1, *argv*[X1\_VALUE], *argv*[Y1\_VALUE]);

    set\_point(&pt2, *argv*[X2\_VALUE], *argv*[Y2\_VALUE]);

    result\_t answer = calculateSlope(&pt1, &pt2, rounding\_place);

    if (answer.error\_code != NO\_ERROR)

    {

        print\_error\_code(answer.error\_code);

        return 1;

    }

    std::cout<<"Slope is: " << answer.value <<'\n';

    return 0;

}

Step 2 Design Test Cases ([test\_slope.py](https://github.com/IntegralWorks/CSCI_4330_Assignment6/blob/main/test_slope.py))

Note on general strategy: all test cases are implemented in Python. The idea is to take advantage of

*int* main(*int* *argc*, *char* \**argv*[])

{

    //input format: <x1> <y1> <x2> <y2> <rounding\_place>

This means we can generate strings in an easier to write language for testing, although we will see how sometimes that is not enough.

from random import randint

import os

from time import sleep

#uncomment to recompile just in case

# os.system('g++ -o slope slope.cpp')

print('-----------------------General Test-----------------------')

general\_test\_record = *dict*()

for i in range(0,3):

    test\_data  = [*f*'{randint(-100, 100)}.{randint(1, 100\_000)}' for i in range(4)]

    test\_data += [randint(3, 10)]

    test\_string = *f*'slope {test\_data[0]} {test\_data[1]} {test\_data[2]} {test\_data[3]} {test\_data[4]}'

    print(*f*"Test String {i}:",test\_string)

    os.system(*f*'slope {test\_data[0]} {test\_data[1]} {test\_data[2]} {test\_data[3]} {test\_data[4]}')

    print('-----------------------')

    general\_test\_record[i] = test\_data

print("Dumping general\_test data:", general\_test\_record)

print('-----------------------End General Test-----------------------\n\n')

print('-----------------------DIV\_BY\_ZERO Test-----------------------')

test\_data    = [*f*'{randint(-100, 100)}.{randint(1, 100\_000)}' for i in range(4)]

test\_data   += [randint(3, 10)]

test\_data[0] = *str*(abs(*float*(test\_data[0])))

test\_data[2] = test\_data[0]

test\_string = *f*'slope {test\_data[0]} {test\_data[1]} {test\_data[2]} {test\_data[3]} {test\_data[4]}'

print(*f*"Test String:",test\_string)

os.system(*f*'slope {test\_data[0]} {test\_data[1]} {test\_data[2]} {test\_data[3]} {test\_data[4]}')

print('-----------------------End DIV\_BY\_ZERO Test-----------------------\n\n')

print('-----------------------INFINITY\_REACHED Test-----------------------')

DBL\_MAX = 1.7976931348623157e+308

DBL\_MIN = 2.2250738585072014e-308

case\_1    = [DBL\_MAX, DBL\_MAX, DBL\_MAX, DBL\_MAX, 16]

case\_2    = [DBL\_MIN, DBL\_MIN, DBL\_MIN, DBL\_MIN, 16]

case\_1\_string = *f*'slope {case\_1[0]} {case\_1[1]} {case\_1[2]} {case\_1[3]} {case\_1[4]}'

case\_2\_string = *f*'slope {case\_2[0]} {case\_2[1]} {case\_2[2]} {case\_2[3]} {case\_2[4]}'

print(*f*"Test case\_1:", case\_1\_string)

os.system(case\_1\_string)

print(*f*"Test case\_2:", case\_2\_string)

os.system(case\_2\_string)

print('-----------------------End INFINITY\_REACHED Test-----------------------\n\n')

print('-----------------------INVALID\_COORD\_INPUT Test-----------------------')

case\_1    = [*f*'{randint(-100, 100)}', *f*'{randint(-100, 100)}', *f*'{randint(-100, 100)}', *f*'{randint(-100, 100)}', randint(3, 10)]

case\_2    = [*f*'lol', *f*'i', *f*'so', *f*'random', randint(3, 10)]

case\_1\_string = *f*'slope {case\_1[0]} {case\_1[1]} {case\_1[2]} {case\_1[3]} {case\_1[4]}'

case\_2\_string = *f*'slope {case\_2[0]} {case\_2[1]} {case\_2[2]} {case\_2[3]} {case\_2[4]}'

print(*f*"Test case\_1:", case\_1\_string)

os.system(case\_1\_string)

print(*f*"Test case\_2:", case\_2\_string)

os.system(case\_2\_string)

print('-----------------------End INVALID\_COORD\_INPUT Test-----------------------\n\n')

print('-----------------------INVALID\_ROUNDING\_PLACE\_INPUT Test-----------------------')

#note: case 2 should be a warning, not an error

cases = ['16', '-14', 'L0L']

for i in range(0,3):

    test\_data  = [*f*'{randint(-100, 100)}.{randint(1, 100\_000)}' for i in range(4)]

    test\_data += [cases[i]]

    test\_string = *f*'slope {test\_data[0]} {test\_data[1]} {test\_data[2]} {test\_data[3]} {test\_data[4]}'

    print(*f*"Test String {i}:",test\_string)

    os.system(*f*'slope {test\_data[0]} {test\_data[1]} {test\_data[2]} {test\_data[3]} {test\_data[4]}')

    print('-----------------------')

    general\_test\_record[i] = test\_data

print('-----------------------End INVALID\_ROUNDING\_PLACE\_INPUT Test-----------------------\n\n')

Breakdown of Deliverables

* Your function implementation in Python, Java, C, or pseudo code. Ensure your code or pseudo code is well documented.
  + All code has been submitted.
* All your test cases, presented systematically, alongside their expected output. Include a rationale for each test case, explaining why it was chosen.
  + Let’s break down the general strategy a touch further and go through it.

First, observe

typedef *enum*

{

    NO\_ERROR,

    DIV\_BY\_ZERO,

    INFINITY\_REACHED,

    INVALID\_COORD\_INPUT,

    INVALID\_ROUNDING\_PLACE\_INPUT

} error\_code\_t;

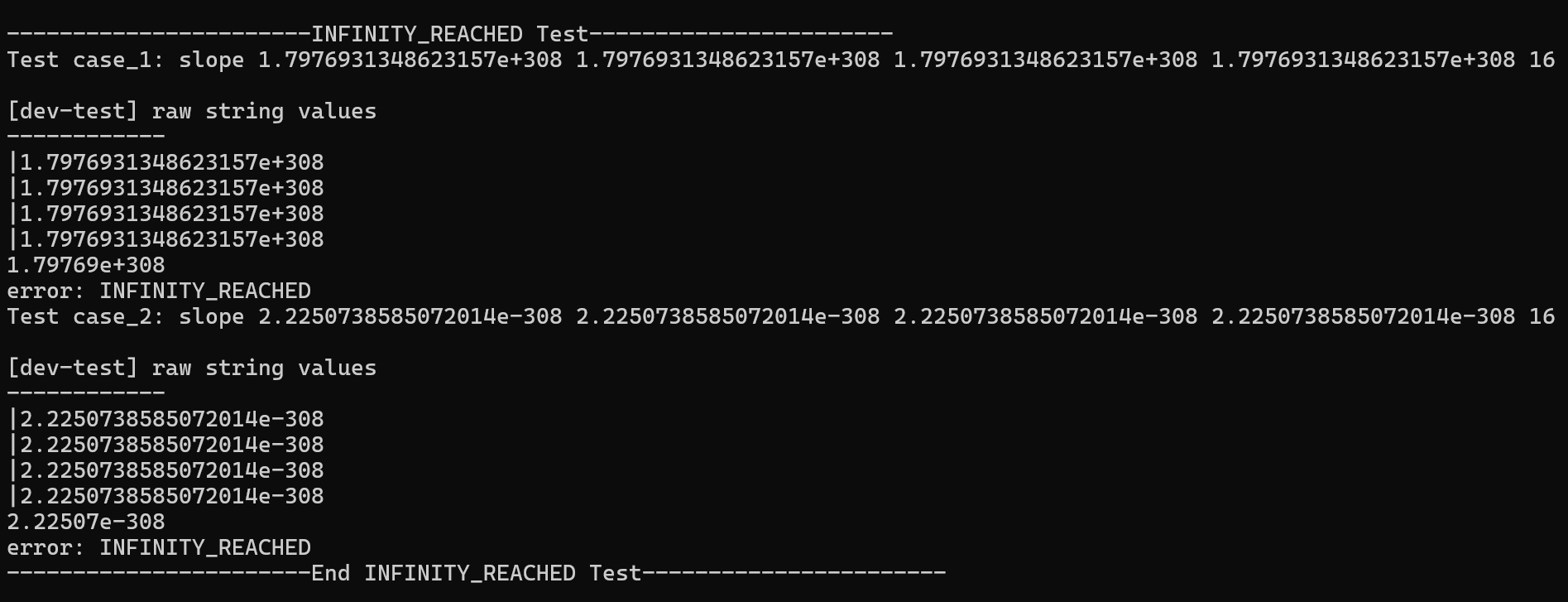
Essentially, this is a terse way of saying, “what could go wrong with the code”? it could:

* Have nothing go wrong (or at least think so)
* Try to divide by zero
* Get too close to the absolute maximum of double floats
* Try to parse bad input
* Try to use an improper rounding value

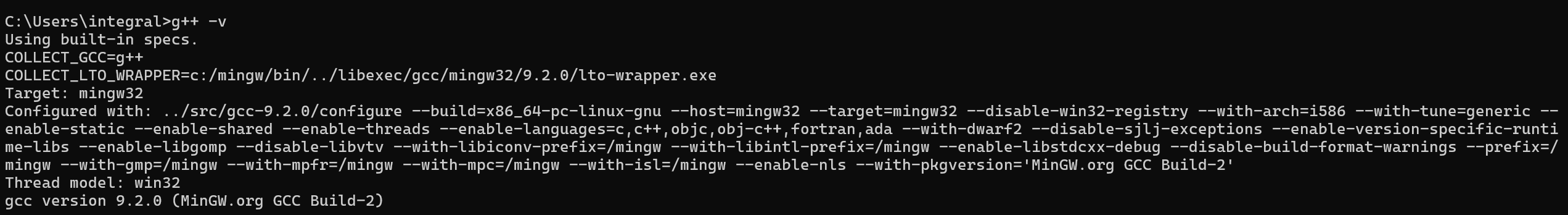
We establish in the Python code a Test for each of these.

* General Test: Mostly serves as an example of what “good” usage looks like.
* DIV\_BY\_ZERO Test: Ensures that x1 and x2 are the same number, meaning they will always cause a 0 denominator.
* INFINITY\_REACHED Test: This was the trickiest and has a well-deserved entry in the “if you encounter any bugs” section. Almost all the rest of the code didn’t really have any challenging bugs. But long story short for now, the idea was to pass DBL\_MAX and DBL\_MIN (constants in C/C++) directly. It didn’t go well. However, we will see that we can learn a lot from this.
* INVALID\_COORD\_INPUT Test: Because of the theory behind regex (more on that in the “Clearly list and explain any assumptions” section), this is more of a sanity check than a mission-critical test.
* INVALID\_ROUNDING\_PLACE\_INPUT Test: Mostly a quick, brisk check of the if (rounding\_place < 0 || rounding\_place >= 16) logic.
* During testing using your test cases, if you encounter any bugs, detail them in the document, specifying how you addressed them.
  + The INFINITY\_REACHED Test did a good job clearly establishing that the code had a bug.

Essentially, “1.7976931348623157e+308” cannot really be passed to the C++ code due to the dependency on **std::stod**. With the dev-tests (tests within the C++ code itself) on, we can see this output



The numerical value is truncated. This was found to be the behavior of **std::stod** with this version of G++ on Windows 10 x86\_64:



C:\Users\integral>g++ -v

Using built-in specs.

COLLECT\_GCC=g++

COLLECT\_LTO\_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/9.2.0/lto-wrapper.exe

Target: mingw32

Configured with: ../src/gcc-9.2.0/configure --build=x86\_64-pc-linux-gnu --host=mingw32 --target=mingw32 --disable-win32-registry --with-arch=i586 --with-tune=generic --enable-static --enable-shared --enable-threads --enable-languages=c,c++,objc,obj-c++,fortran,ada --with-dwarf2 --disable-sjlj-exceptions --enable-version-specific-runtime-libs --enable-libgomp --disable-libvtv --with-libiconv-prefix=/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --disable-build-format-warnings --prefix=/mingw --with-gmp=/mingw --with-mpfr=/mingw --with-mpc=/mingw --with-isl=/mingw --enable-nls --with-pkgversion='MinGW.org GCC Build-2'

Thread model: win32

gcc version 9.2.0 (MinGW.org GCC Build-2)

aThis is the reason for this somewhat convoluted, “magic number” looking scrawl (it probably would have been better to make an enum={DBL\_MAX\_TRUNCATED=1.79769e+308, DBL\_MIN\_TRUNCATED=2.22507e-308}:

if (std::regex\_search(value\_strs[i], float\_pattern))

{

    if ( (abs(std::stod(value\_strs[i])) >= 1.79769e+308) || (abs(std::stod(value\_strs[i])) <= 2.22507e-308) )

    {

        std::cout<<std::stod(value\_strs[i])<<'\n';

        return INFINITY\_REACHED;

    }

}

(emphasis: if ( (abs(std::stod(value\_strs[i])) >= 1.79769e+308) || (abs(std::stod(value\_strs[i])) <= 2.22507e-308) ))

Usage of abs is because 2.22507e-308 is a (tiny) positive number, and while technically it would have been better to check for a positive or negative and then do the comparison, this is something of an edge case so abs felt fine to use.

The moral of the story is testing “in the code” and “out of the code” are both important.

Any other bugs during development were either trivial or not really discovered by the tests.

One thing, when testing code in one language with another, consider doing things like this

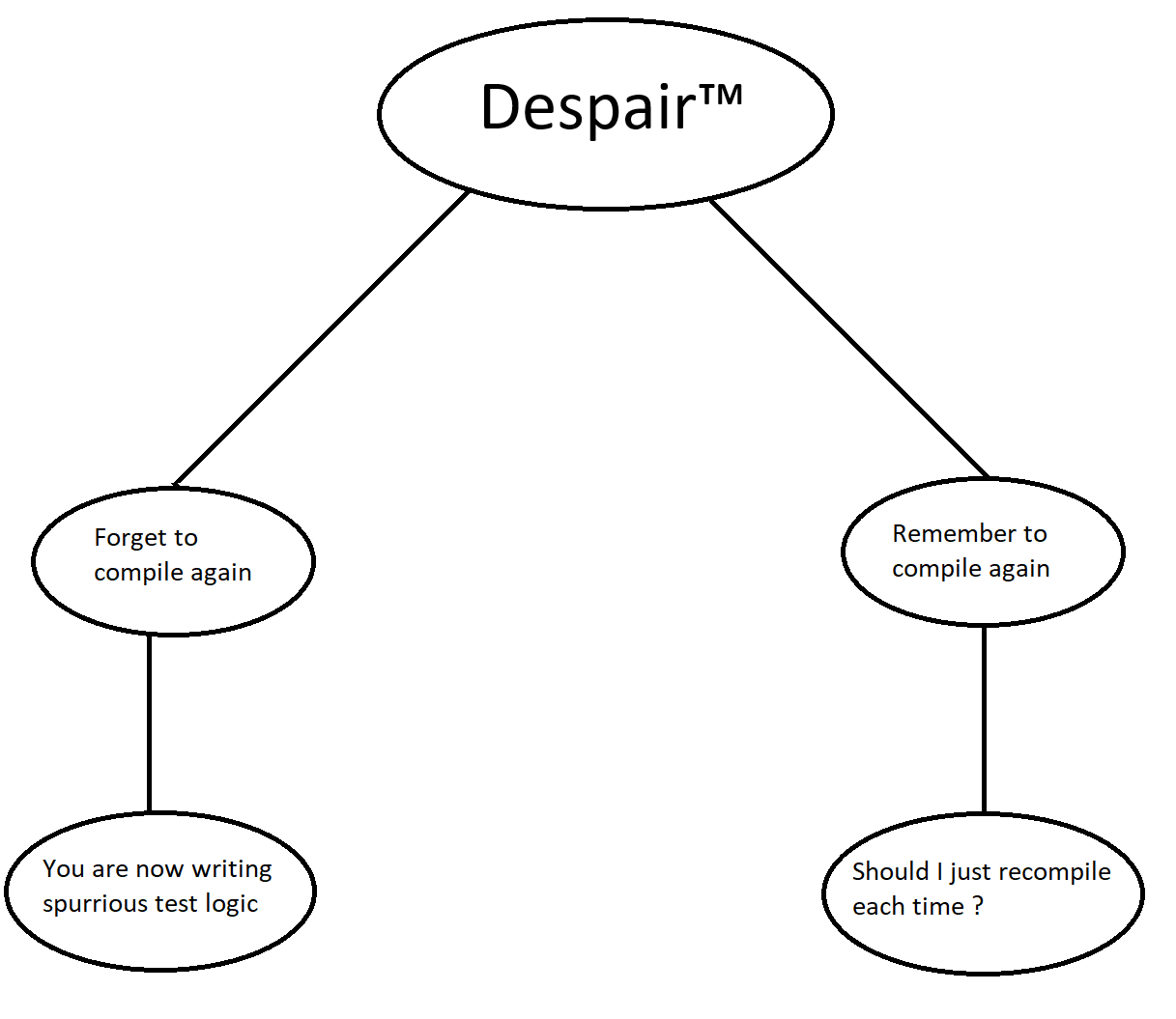
#uncomment to recompile just in case

os.system('g++ -o slope slope.cpp')

The reason is because the tests eventually become sophisticated programs with their own logic, and you will find yourself writing in both languages at once. It also doesn’t help that C/C++ is compiled and Python is interpreted (i.e. doesn’t require compilation**\***) and so it’s easy to

* writing new C/C++ code
* write the corresponding Python test code
* ????
* Despair™

Despair™ has something of a formal definition:



Aside: C/C++ has an aide to make recompilation less painful (and testing + cross-compilation/multi-versioning more granular): #ifdef/#ifndef. See: <https://learn.microsoft.com/en-us/cpp/preprocessor/hash-ifdef-and-hash-ifndef-directives-c-cpp?view=msvc-170>

The idea is that you can cordon off sections of the code and have them compile or not compile depending on arbitrary flags.

**\***Note: I found out this is not quite true when using pyinstaller (i.e. making binaries with python) but that is not relevant to the assignment. However, for the general topic of Software Engineering [in Python] it is essential to know about   
python -m compileall <dir> (<https://docs.python.org/3.1/library/compileall.html>)

And how pyinstaller works in general (ex. <https://pyinstaller.org/en/stable/spec-files.html>  
(I just wanted to note that “python is never compiled” is not quite a true statement. I meant it’s “never compiled” in the sense of updating the code versus say C/C++.)

* Clearly list and explain any assumptions you made regarding the function's behavior or input conditions.

Here are some assumptions about how the code works and how said assumption was handled.

* If this code were a production product, it would have a strict specification about input. Input in any form aside from <x1> <y1> <x2> <y2> <rounding\_place> should be regarded as an auto-fail state with 0 leeway.
  + std::regex float\_pattern(R"([0-9]+\.[0-9]+(?:e[+-]?[0-9]{1,308})?)"); is meant to handle the bulk of this. The regex was tested before development. Regex is straightforward to test with a theoretical background of Regular Expressions (although not necessarily “easy”).
    - Aside: even though Regular Expressions (the Computer Science concept) are highly sound and well-defined, ironically **Regex** is implemented in many different ways, so the biggest challenge with Regex is reconciling how there is a Perl Regex, a Python Regex, a C regex, a C++ std Regex, a C++ Boost Regex, a C++ <…> Regex … ad nauseum
    - Luckily the pattern of scientific numbers isn’t too difficult[, unlike say HTML.](https://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags)
* Positive or negative numbers above 1.79769e+308 or below 2.22507e-308 are considered invalid and wholly unsupported.
  + (This was elaborated on in the previous section)
* All **std** functions like **std::stoi** and **std::stod** can throw exceptions and while error\_code\_t values were not made for those, they are assumed to be valid failure states for the code.
  + It was decided that doing this code in Python would not be novel enough for study, and doing it in C would require a lot of cruft. C++ was a good middle-ground.