Write a C program to determine whether a X by Y rectangle can fit within a square of side S, and if so, print the minimum angle it has to be rotated by in order to fit.

A rectangle of width X and height Y fits in a square of side S at some rotation θ iff there exists $\theta \in [0, \pi/2]$ such that

$$X\cos\theta + Y\sin\theta \le S \quad \text{and} \quad X\sin\theta + Y\cos\theta \le S, \; [][]$$

and the minimum such θ can be found by a 1D search. [1] [2]

Below is a simple C program using doubles and a numeric search (binary search on θ) that:

- Reads X, Y, S.
- Checks if it can fit at all.
- If it can, finds the minimum angle in degrees to 1e-6 precision.

```
#include <stdio.h>
#include <math.h>

static int can_fit(double w, double h, double s, double theta)
{
    double c = cos(theta);
    double t = sin(theta);

    /* Bounding box of rotated rectangle about its center:
       width'  = w * fabs(c) + h * fabs(t)
       height' = w * fabs(t) + h * fabs(c)  [web:9] */
    double bw = w * fabs(c) + h * fabs(t);
    double bh = w * fabs(t) + h * fabs(c);

    return (bw <= s + 1e-12 && bh <= s + 1e-12);
}

int main(void)
{
    double X, Y, S;

    if (scanf("%lf %lf %lf", &X, &Y, &S) != 3) {
        fprintf(stderr, "Usage: X Y S\n");
        return 1;
    }
```

```
    /* Normalize so X >= Y, square is symmetric. */
    if (Y > X) {
        double tmp = X;
        X = Y;
        Y = tmp;
    }

    /* Trivial checks. */
    if (X <= S && Y <= S) {
        printf("Fits without rotation (angle = 0 degrees)\n");
        return 0;
    }

    /* If even at 45 degrees (pi/4) it doesn't fit, then no angle works. */
    if (!can_fit(X, Y, S, M_PI / 4.0)) {
        printf("Does not fit in the square for any rotation.\n");
        return 0;
    }

    /* Binary search smallest angle in [0, pi/4] that fits. [web:11][web:13] */
    double lo = 0.0;
    double hi = M_PI / 4.0;
    for (int i = 0; i < 80; ++i) { /* enough for ~1e-12 rad precision */
        double mid = 0.5 * (lo + hi);
        if (can_fit(X, Y, S, mid))
            hi = mid;
        else
            lo = mid;
    }

    double theta = hi;                      /* radians */
    double degrees = theta * 180.0 / M_PI;  /* to degrees */

    printf("Fits. Minimum rotation angle: %.6f degrees\n", degrees);
    return 0;
}
```

How it works

- The function `can_fit` computes the axis-aligned bounding box of the rectangle after rotation by θ about its center, which has side lengths $w\cos\theta + h\sin\theta$ and $w\sin\theta + h\cos\theta$ up to absolute values.[1]
- The rectangle fits in the square iff both of those are $\leq S$.[2]
- Because the configuration is symmetric, the minimum angle lies in $[0, \pi/4]$ if any angle works, so a binary search on that interval finds the smallest θ that satisfies the constraints.[3] [4]

⁂

Can't it be solved using pythagoras' theorem without the iterative search?

For general X, Y, S there is no single Pythagoras-only closed-form for the *minimum* angle; you end up solving trig equations whose solution is transcendental, so numeric methods are standard.[19] [20]

What Pythagoras gives

- Pythagoras gives relations like $d^2 = X^2 + Y^2$ for the rectangle diagonal and $S^2 = S^2 + 0^2$ for the square side, and lets you compare the diagonal d with S.[21] [22]
- That is enough to answer "does it fit if aligned with a diagonal?" but not "what is the smallest rotation where the bounding box just equals S × S?".[23] [19]

Why the angle needs trig

- The bounding box of a rotated X by Y rectangle is
  - width: $X\cos\theta + Y\sin\theta$
  - height: $X\sin\theta + Y\cos\theta$.[24] [19]
- The "just fits" condition is solving $X\cos\theta + Y\sin\theta = S$ and $X\sin\theta + Y\cos\theta = S$ together, which reduces to nonlinear equations in $\sin\theta$ and $\cos\theta$, not a single right triangle where Pythagoras alone applies.[25] [20]

When closed forms appear

- In special symmetric cases (e.g., X = Y or specific aspect ratios) the minimum angle may be 0, 45°, or given by a simple arctan expression derived once you already used trig identities.[20] [19]
- For arbitrary X, Y, S, practical treatments use numeric search (binary search, Newton, etc.) over $\theta \in [0, \pi/4]$, which is robust and cheap in code.[19] [20]

⁂

Then modify the requirement such that a 45 degree angle is used when it does not fit axis-aligned, then move the object to minimise X and Y for the bounding box of the rotated rectangle.

Yes; with your new requirement the geometry becomes simple, and you only need the 45° case plus a translation of the rotated rectangle so that its axis-aligned bounding box is minimized and centered.

Geometry at 45 degrees

For a rectangle X by Y, rotated by 45° about its center, the axis-aligned bounding box sides are

$$W = (X + Y)\frac{\sqrt{2}}{2}, \quad H = (X + Y)\frac{\sqrt{2}}{2}.[][]$$

So the bounding box is a square, and the rotated rectangle fits inside the square of side S if and only if

$$(X + Y)\frac{\sqrt{2}}{2} \leq S.[]$$

Placement to minimize X and Y

- If you rotate about the rectangle center and then translate so that the bounding box is centered in the square, the bounding box extents are $\pm W/2$ and $\pm H/2$; this is the minimal possible axis-aligned bounding box for that orientation.[39] [40]
- There is no further translation that can reduce those W, H values; translating just shifts min/max but does not change width/height.[41] [39]

C code sketch

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    double X, Y, S;
    const double SQRT2_OVER_2 = M_SQRT1_2; /* sqrt(2)/2 */

    if (scanf("%lf %lf %lf", &X, &Y, &S) != 3) {
        fprintf(stderr, "Usage: X Y S\n");
        return 1;
    }

    /* Axis-aligned fit first. */
    if (X <= S && Y <= S) {
        printf("Fits axis-aligned (angle = 0 degrees).\n");
        return 0;
    }

    /* 45-degree rotated bounding box size. */
    double W = (X + Y) * SQRT2_OVER_2;  /* width == height at 45deg */

    if (W <= S) {
        /* Fits at 45 degrees; place centered in the square. */
        double angle_deg = 45.0;
        double bx = (S - W) * 0.5;  /* bounding box min x */
        double by = (S - W) * 0.5;  /* bounding box min y */
        printf("Fits at 45 degrees.\n");
        printf("Bounding box size: %.6f x %.6f\n", W, W);
        printf("Place bounding box with min corner at (%.6f, %.6f)\n",
```

```
                bx, by);
        printf("Rotation angle: %.6f degrees\n", angle_deg);
    } else {
        printf("Does not fit even at 45 degrees.\n");
    }

    return 0;
}
```

- This uses the closed-form 45° bounding box formula; no iteration is needed.[40]
- The “minimize X and Y” is achieved by using the smallest possible axis-aligned bounding box at that angle (about the center) and then centering it in the square.[39] [40]

⁂

modify that solution to write a command-line program with parameters using gnu getopt parameter parsing for --x --y and --s 'long style' parameters (even though they are single-letter options in practice)

Below is a self-contained command-line program that uses `getopt_long` to parse `--x`, `--y`, and `--s` (also accepting `-x`, `-y`, `-s`) and then applies the “0° or 45° with centered bounding box” logic.

Program behavior

```
- Usage: `rectfit --x <width> --y <height> --s <square_side>`
```

- If X,Y fit axis-aligned within S, prints angle 0.
- Else checks 45° bounding box and, if it fits, prints angle 45°, bounding box size, and placement.
- Otherwise reports that it does not fit.

C source using getopt_long

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <math.h>

static void usage(const char *prog)
{
    fprintf(stderr,
        "Usage: %s --x <width> --y <height> --s <square_side>\n"
        "       %s -x <width> -y <height> -s <square_side>\n",
        prog, prog);
```

```
}

/* Returns 0 on success, nonzero on error. */
int main(int argc, char *argv[])
{
    double X = 0.0, Y = 0.0, S = 0.0;
    int have_x = 0, have_y = 0, have_s = 0;

    static struct option long_opts[] = {
        { "x", required_argument, 0, 'x' },
        { "y", required_argument, 0, 'y' },
        { "s", required_argument, 0, 's' },
        { "help", no_argument, 0, 'h' },
        { 0, 0, 0, 0 }
    };

    int opt;
    int opt_index = 0;

    while ((opt = getopt_long(argc, argv, "x:y:s:h", long_opts, &opt_index)) != -1) {
        switch (opt) {
        case 'x':
            X = atof(optarg);
            have_x = 1;
            break;
        case 'y':
            Y = atof(optarg);
            have_y = 1;
            break;
        case 's':
            S = atof(optarg);
            have_s = 1;
            break;
        case 'h':
            usage(argv[^4_0]);
            return 0;
        default:
            usage(argv[^4_0]);
            return 1;
        }
    }

    if (!have_x || !have_y || !have_s) {
        usage(argv[^4_0]);
        return 1;
    }

    if (X <= 0.0 || Y <= 0.0 || S <= 0.0) {
        fprintf(stderr, "All dimensions must be positive.\n");
        return 1;
    }

    /* Axis-aligned check. */
    if (X <= S && Y <= S) {
        printf("Fits axis-aligned.\n");
        printf("Rotation angle: 0.0 degrees\n");
```

```
        printf("Bounding box size: %.6f x %.6f\n", X, Y);
        printf("Place bounding box with min corner at (%.6f, %.6f)\n",
               (S - X) * 0.5, (S - Y) * 0.5);
        return 0;
    }

    /* 45-degree case. Bounding box is a square of side (X + Y)*sqrt(2)/2. [web:49][web:5
    const double SQRT2_OVER_2 = M_SQRT1_2; /* sqrt(2)/2 */
    double W = (X + Y) * SQRT2_OVER_2;

    if (W <= S) {
        double angle_deg = 45.0;
        double bx = (S - W) * 0.5;
        double by = (S - W) * 0.5;

        printf("Fits at 45 degrees.\n");
        printf("Rotation angle: %.6f degrees\n", angle_deg);
        printf("Bounding box size: %.6f x %.6f\n", W, W);
        printf("Place bounding box with min corner at (%.6f, %.6f)\n",
               bx, by);
        return 0;
    }

    printf("Does not fit axis-aligned or at 45 degrees.\n");
    return 0;
}
```

- `getopt_long` is declared in `<getopt.h>` and is used here with a `struct option` array defining the `--x`, `--y`, `--s`, and `--help` long options, mapped to short options `x`, `y`, `s`, `h`.[58] [59]
- The 45° bounding box formula `(X + Y)*sqrt(2)/2` gives the minimal axis-aligned box size for that orientation, which is then centered in the square to minimize the X and Y extents.[60] [61]

⁂

Modify that program so that if only one of x or y is given, the maximum value of the other dimension is determined and printed. s must be given.

Yes; with your 0°/45° rule this is straightforward algebra plus some option-parsing logic. If only one of X or Y is given, the other is solved from the relevant inequality and printed.

- Axis-aligned fit: $X \le S$ and $Y \le S$.
- 45° fit: $(X+Y)\frac{\sqrt{2}}{2} \le S \;\Rightarrow\; X+Y \le S\sqrt{2}$.[80]

Below is an updated program:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
```

```
#include <math.h>

static void usage(const char *prog)
{
    fprintf(stderr,
        "Usage:\n"
        "  %s --x <width> --y <height> --s <square_side>\n"
        "  %s --x <width> --s <square_side>\n"
        "  %s --y <height> --s <square_side>\n"
        "Short forms: -x, -y, -s\n",
        prog, prog, prog);
}

int main(int argc, char *argv[])
{
    double X = 0.0, Y = 0.0, S = 0.0;
    int have_x = 0, have_y = 0, have_s = 0;

    static struct option long_opts[] = {
        { "x",    required_argument, 0, 'x' },
        { "y",    required_argument, 0, 'y' },
        { "s",    required_argument, 0, 's' },
        { "help", no_argument,       0, 'h' },
        { 0, 0, 0, 0 }
    };

    int opt;
    int opt_index = 0;

    while ((opt = getopt_long(argc, argv, "x:y:s:h", long_opts, &opt_index)) != -1) {
        switch (opt) {
        case 'x':
            X = atof(optarg);
            have_x = 1;
            break;
        case 'y':
            Y = atof(optarg);
            have_y = 1;
            break;
        case 's':
            S = atof(optarg);
            have_s = 1;
            break;
        case 'h':
            usage(argv[^5_0]);
            return 0;
        default:
            usage(argv[^5_0]);
            return 1;
        }
    }

    if (!have_s) {
        fprintf(stderr, "Error: --s <square_side> is required.\n");
        usage(argv[^5_0]);
        return 1;
```

```
    }

    if (S <= 0.0) {
        fprintf(stderr, "Error: square side must be positive.\n");
        return 1;
    }

    /* Case A: only X is given, compute max Y. */
    if (have_x && !have_y) {
        if (X <= 0.0) {
            fprintf(stderr, "Error: X must be positive.\n");
            return 1;
        }

        /* Axis-aligned limit: Y <= S. */
        double y_axis = S;

        /* 45-degree limit: X + Y <= S*sqrt(2). [web:17] */
        double y_rot = S * M_SQRT2 - X;

        double y_max;
        int use_axis = 0, use_rot = 0;

        if (y_axis > 0.0)
            use_axis = 1;
        if (y_rot > 0.0)
            use_rot = 1;

        if (!use_axis && !use_rot) {
            printf("No positive Y fits for X = %.6f and S = %.6f.\n", X, S);
            return 0;
        }

        if (use_axis && use_rot)
            y_max = (y_axis < y_rot) ? y_axis : y_rot;
        else if (use_axis)
            y_max = y_axis;
        else
            y_max = y_rot;

        printf("Given X = %.6f, S = %.6f\n", X, S);
        printf("Maximum Y that fits (0° or 45°): %.6f\n", y_max);
        return 0;
    }

    /* Case B: only Y is given, compute max X. */
    if (!have_x && have_y) {
        if (Y <= 0.0) {
            fprintf(stderr, "Error: Y must be positive.\n");
            return 1;
        }

        /* Axis-aligned limit: X <= S. */
        double x_axis = S;

        /* 45-degree limit: X + Y <= S*sqrt(2). [web:17] */
```

```
        double x_rot = S * M_SQRT2 - Y;

        double x_max;
        int use_axis = 0, use_rot = 0;

        if (x_axis > 0.0)
            use_axis = 1;
        if (x_rot > 0.0)
            use_rot = 1;

        if (!use_axis && !use_rot) {
            printf("No positive X fits for Y = %.6f and S = %.6f.\n", Y, S);
            return 0;
        }

        if (use_axis && use_rot)
            x_max = (x_axis < x_rot) ? x_axis : x_rot;
        else if (use_axis)
            x_max = x_axis;
        else
            x_max = x_rot;

        printf("Given Y = %.6f, S = %.6f\n", Y, S);
        printf("Maximum X that fits (0° or 45°): %.6f\n", x_max);
        return 0;
    }

    /* Case C: both X and Y given -> original behavior. */
    if (!have_x || !have_y) {
        fprintf(stderr, "Error: must supply either (X and Y) or exactly one of them.\n");
        usage(argv[^5_0]);
        return 1;
    }

    if (X <= 0.0 || Y <= 0.0) {
        fprintf(stderr, "Error: X and Y must be positive.\n");
        return 1;
    }

    /* Axis-aligned check. */
    if (X <= S && Y <= S) {
        printf("Fits axis-aligned.\n");
        printf("Rotation angle: 0.0 degrees\n");
        printf("Bounding box size: %.6f x %.6f\n", X, Y);
        printf("Place bounding box with min corner at (%.6f, %.6f)\n",
               (S - X) * 0.5, (S - Y) * 0.5);
        return 0;
    }

    /* 45-degree case: bounding box side W = (X + Y)*sqrt(2)/2. [web:17] */
    {
        const double SQRT2_OVER_2 = M_SQRT1_2;
        double W = (X + Y) * SQRT2_OVER_2;

        if (W <= S) {
            double angle_deg = 45.0;
```

```
            double bx = (S - W) * 0.5;
            double by = (S - W) * 0.5;

            printf("Fits at 45 degrees.\n");
            printf("Rotation angle: %.6f degrees\n", angle_deg);
            printf("Bounding box size: %.6f x %.6f\n", W, W);
            printf("Place bounding box with min corner at (%.6f, %.6f)\n",
                   bx, by);
            return 0;
        }
    }

    printf("Does not fit axis-aligned or at 45 degrees.\n");
    return 0;
}
```

- The “max other dimension” comes from intersecting the axis-aligned constraint (other side ≤ S) with the 45° constraint $X + Y \le S\sqrt{2}$, and taking the smaller positive bound.[81] [80]
- `getopt_long` handling is unchanged in structure; only the post-parse logic branches on which of `--x`/`--y` were supplied.[82] [83]

⁂

1. https://stackoverflow.com/questions/26856020/calculating-smallest-rotated-rectangle-that-bounds-another-and-maintains-aspect
2. https://stackoverflow.com/questions/5789239/calculate-largest-inscribed-rectangle-in-a-rotated-rectangle
3. http://answers.google.com/answers/threadview/id/771496.html
4. https://www.reddit.com/r/askmath/comments/14x43bp/modeling_the_relationship_between_a_bounding_box/
5. https://www.reddit.com/r/LinearAlgebra/comments/1gx7e8k/draw_rotated_bounding_rectangle/
6. https://www.physicsforums.com/threads/how-to-calc-sides-of-rectangle-inside-a-rotated-rectangle.465070/
7. https://discourse.mcneel.com/t/randomly-rotate-the-rectangle-by-0-or-90-degrees/193323
8. https://www.youtube.com/watch?v=YSqDalSxm1c
9. https://www.youtube.com/watch?v=MvlhMEE9zuc
10. https://www.mathworks.com/matlabcentral/answers/1650320-rotate-a-rectangle-within-boundaries
11. https://forum.lazarus.freepascal.org/index.php?topic=43251.0
12. https://www.youtube.com/watch?v=d8VfYyBlHAI
13. https://forums.autodesk.com/t5/inventor-forum/problem-rotating-a-rectangle-in-a-sketch/td-p/8540546
14. https://structx.com/Shape_Formulas_033.html
15. https://www.freemathhelp.com/forum/threads/find-smallest-area-of-rectangle-based-on-lines-inside.137699/
16. https://www.reddit.com/r/askmath/comments/pz1w0y/formula_to_determine_size_of_rectangle_based_on/

17. https://stackoverflow.com/questions/13002979/how-to-calculate-rotation-angle-from-rectangle-points
18. https://www.youtube.com/watch?v=U975eMcp2Xc
19. https://stackoverflow.com/questions/5789239/calculate-largest-inscribed-rectangle-in-a-rotated-rectangle
20. https://matchmaticians.com/questions/bh797r/calculating-aspect-ratio-limits-of-rotated-rectangle-within
21. https://en.wikipedia.org/wiki/Pythagorean_theorem
22. https://www.grc.nasa.gov/www/BGH/pythag.html
23. https://www.bbc.co.uk/bitesize/articles/zgf8ng8
24. https://structx.com/Shape_Formulas_033.html
25. http://answers.google.com/answers/threadview/id/771496.html
26. https://central.edu/writing-anthology/2019/01/31/159/
27. https://helpingwithmath.com/pythagoras-theorem/
28. https://community.claris.com/en/s/question/0D50H00007NItCZSA1/calculating-how-many-rectangles-fit-within-a-larger-rectangle
29. https://www.engineeringtoolbox.com/smaller-rectangles-within-larger-rectangle-d_2111.html
30. https://www.reddit.com/r/explainlikeimfive/comments/u2saws/eli5_in_pythagoras_theorem_why_must_the_length_of/
31. https://www.youtube.com/watch?v=6f19u7eYIPA
32. https://www.youtube.com/watch?v=nDNUqKIfW_w
33. https://www.youtube.com/watch?v=_K-iTgAE09Y
34. https://www.reddit.com/r/algorithms/comments/ojno5c/placing_rectangles_within_a_larger_rectangle/
35. https://www.physicsforums.com/threads/how-to-calc-sides-of-rectangle-inside-a-rotated-rectangle.465070/
36. https://www.mathworks.com/matlabcentral/answers/304942-how-to-rotate-a-rectangle-until-one-of-the-edges-is-parallel-to-the-x-axis-with-unknown-rotation-an
37. https://www.youtube.com/watch?v=Y5wiWCR9Axc
38. https://www.cut-the-knot.org/pythagoras/
39. https://stackoverflow.com/questions/3231176/how-to-get-size-of-a-rotated-rectangle
40. https://www.asawicki.info/news_1376_calculating_aabb_of_a_rotated_2d_rectangle
41. https://www.html5gamedevs.com/topic/685-anyone-know-how-to-get-the-bounding-box-of-a-rotated-rectangle/
42. https://www.reddit.com/r/askmath/comments/14x43bp/modeling_the_relationship_between_a_bounding_box/
43. https://pyrr.readthedocs.io/en/latest/api_aabb.html
44. https://forums.oracle.com/ords/apexds/post/find-new-width-and-height-of-a-rotated-item-1465
45. https://www.reddit.com/r/opengl/comments/rexsn6/help_with_aabb_rotation/
46. https://devforum.roblox.com/t/how-do-i-get-a-rotated-bounding-box-between-two-randomly-rotated-parts/2096998
47. https://developer.nvidia.com/blog/detecting-rotated-objects-using-the-odtk/
48. https://github.com/leimao/Rotated-Rectangle-Crop-OpenCV/blob/master/rotated_rect_crop.py

49. https://www.youtube.com/watch?v=MvlhMEE9zuc
50. https://krita-artists.org/t/is-it-possible-to-rotate-the-free-transform-handle-box-without-actually-warping-the-contents/18613
51. https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection
52. https://stackoverflow.com/questions/622140/calculate-bounding-box-coordinates-from-a-rotated-rectangle
53. https://community.monogame.net/t/how-would-i-go-about-making-a-rotated-bounding-box-rectangle-as-well-as-getting-the-displacement-vector-for-the-intersecting-object/17148
54. https://www.gamedev.net/forums/topic/540248-calculating-minimun-bounding-box-of-a-2d-rotated-rectangle/
55. https://alpha.inkscape.org/vectors/www.inkscapeforum.com/viewtopic6e11.html?t=9817
56. https://blog.unitlab.ai/guide-to-rotating-bounding-boxes/
57. https://docs.humansignal.com/templates/image_bbox
58. https://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Option-Example.html
59. http://www.gnu.org/s/libc/manual/html_node/Getopt-Long-Options.html
60. https://stackoverflow.com/questions/3231176/how-to-get-size-of-a-rotated-rectangle
61. https://www.asawicki.info/news_1376_calculating_aabb_of_a_rotated_2d_rectangle
62. https://azrael.digipen.edu/~mmead/www/mg/getopt/index.html
63. https://docs.oracle.com/cd/E86824_01/html/E54766/getopt-long-3c.html
64. https://refspecs.linuxbase.org/LSB_5.0.0/LSB-Core-generic/LSB-Core-generic/baselib-getopt-long-3.html
65. https://stackoverflow.com/questions/7489093/getopt-long-proper-way-to-use-it
66. https://docs.mojolicious.org/Getopt/Long
67. https://www.ibm.com/docs/en/zos/3.1.0?topic=functions-getopt-long-command-long-option-parsing
68. https://www.geeksforgeeks.org/c/getopt-function-in-c-to-parse-command-line-arguments/
69. https://stackoverflow.com/questions/35805020/c-getopt-long-example-tests-in-case-0
70. https://azrael.digipen.edu/~mmead/www/Courses/CS180/getopt.html
71. https://gist.github.com/ashwin/d88184923c7161d368a9
72. http://ld2016.scusa.lsu.edu/gnu_c_libs/Getopt-Long-Options.html
73. http://wiki.call-cc.org/egg/getopt-long
74. https://leimao.github.io/blog/Argument-Parser-Getopt-C/
75. https://courses.cs.duke.edu/spring04/cps108/resources/getoptman.html
76. https://manual.cs50.io/3/getopt_long
77. https://www.cs.wm.edu/~smherwig/courses/csci415-common/parse-options/index.html
78. https://man.openbsd.org/getopt_long.3
79. https://stackoverflow.com/questions/402377/using-getopts-to-process-long-and-short-command-line-options
80. https://stackoverflow.com/questions/5789239/calculate-largest-inscribed-rectangle-in-a-rotated-rectangle

81. https://www.physicsforums.com/threads/rectangle-inscribed-in-another-rectangle-solutions-for-all-cases.508715/
82. https://refspecs.linuxbase.org/LSB_5.0.0/LSB-Core-generic/LSB-Core-generic/baselib-getopt-long-3.html
83. https://www.gnu.org/software/libc/manual/html_node/Getopt-Long-Option-Example.html
84. https://docs.mojolicious.org/Getopt/Long
85. https://www.engineeringtoolbox.com/smaller-rectangles-within-larger-rectangle-d_2111.html
86. https://www.youtube.com/watch?v=prx_Bf2hakw
87. https://math.libretexts.org/Bookshelves/Algebra/Advanced_Algebra/03:_Solving_Linear_Systems/3.07:_Solving_Systems_of_Inequalities_with_Two_Variables
88. https://cfengine.com/blog/2021/optional-arguments-with-getopt-long/
89. https://alpha.inkscape.org/vectors/www.inkscapeforum.com/viewtopic9345.html?t=15777
90. https://www.wolframalpha.com/calculators/equation-solver-calculator
91. https://www.symbolab.com/solver/equation-calculator
92. https://www.mathpapa.com/algebra-calculator.html
93. https://www.youtube.com/watch?v=0IFPixtQpYM
94. https://manual.cs50.io/3/getopt_long
95. https://www.symbolab.com/solver/algebra-calculator
96. https://www.reddit.com/r/askmath/comments/8bzyzf/what_is_the_maximum_possible_side_of_a_rectangle/
97. https://www.facebook.com/groups/mathpuz/posts/2954950131347476/
98. https://www.youtube.com/watch?v=7YxINrFO01M
99. https://www.omnicalculator.com/math/square-root
100. https://www.reddit.com/r/askmath/comments/vo7s4e/anyone_know_how_to_solve_this_questions_the/