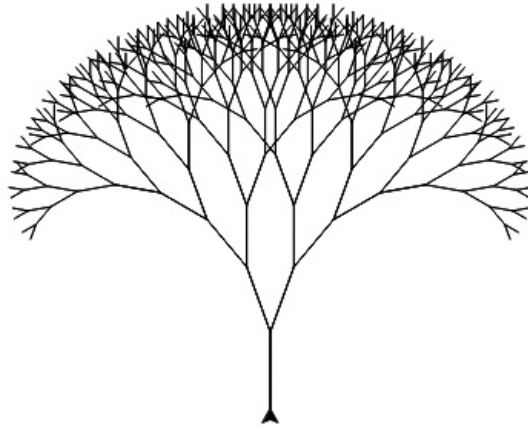


Lab 1

Lab 1 - Recursive Tree



Recursive Tree

Trees can be drawn recursively. Draw a branch. At the end of the branch, draw two smaller branches with one to the left and the other to the right. Repeat until a certain condition is true. This program will walk you through drawing a tree in this way.

Turtle Graphics Review

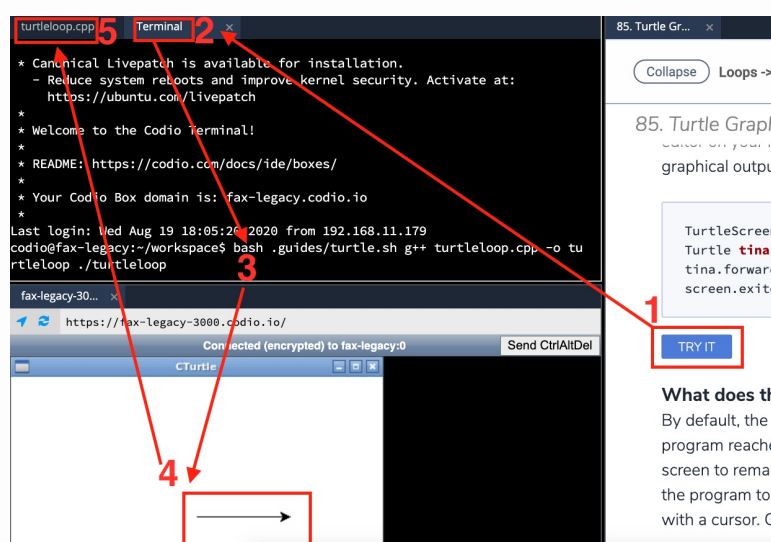
We will be using Turtle Graphics to visualize recursion. Here is some review on what a Turtle object (tina) can do in C++:

▼ *Turtle Graphics Review*

- `tina.forward(n)` - Where `n` represents the number of pixels.
 - `tina.backward(n)` - Where `n` represents the number of pixels.
 - `tina.right(d)` - Where `d` represents the number of degrees.
 - `tina.left(d)` - Where `d` represents the number of degrees.
 - `tina.pencolor({"COLOR"})` - Where `COLOR` represents the track or line color you want tina to leave behind.
 - `tina.width(W)` - Where `W` represents how wide (in pixels) tina's track is.
 - `tina.shape("SHAPE")` - Where `SHAPE` represents the shape tina takes.
 - `tina.speed(SPEED)` - Where `SPEED` represents how fast tina moves
-

Command	Parameter	Examples
<code>tina.pencolor({"COLOR"})</code>	Where COLOR represents the track or line color you want tina to leave behind	red, orange, yellow, green, blue, purple
<code>tina.width(W)</code>	Where w represents how wide (in pixels) tina's track is	any positive integer (e.g. 1, 10, 123, etc.)
<code>tina.shape("SHAPE")</code>	Where SHAPE represents the shape tina takes	triangle, indented triangle, square, arrow
<code>tina.speed(SPEED)</code>	Where SPEED represents how fast tina moves	TS_FASTEST, TS_FAST, TS_NORMAL, TS_SLOW, TS_SLOWEST

And some instructions on how Turtle Graphics work on the Terminal:



.guides/img/TurtleGraphicsFlow

1. TRY IT button is clicked by the user.
2. The Terminal tab is opened.
3. The terminal runs the command to compile the program and to display the graphical output.
4. The output is displayed as a screen on the bottom left panel. You can click the screen to close the output.
5. Click on the file name tab to go back to the text editor if you want to make changes to the program.

Program Instructions

Let's start by creating a canvas screen and a Turtle object tina in `main()` to allow the Turtle object to move around on. Feel free to edit the screen size so that it fits comfortably on your monitor. You'll also notice that there is function called `RecursiveTree()`. This function takes three parameters, `branch_length`, `angle`, and `t`.

Note that when passing an object (like a Turtle), you should pass it as a reference using the `&` symbol (i.e. `Turtle& t`).

This is the code you have so far in the text editor:

```
////////// DO NOT EDIT HEADER! //////////
#include <iostream>
#include "CTurtle.hpp"
#include "CImg.h"
using namespace cturtle;
using namespace std;
////////////////////////////////////////

/**
 * @param branch_length An integer
 * @param angle The angle of degree
 * @param t A Turtle object
 * @return A drawing symbolizing a tree branch
 */
void RecursiveTree(int branch_length, int angle, Turtle& t) {

    //add function definitions below

    //add function definitions above

}

int main(int argc, char** argv) {

    //add code below this line

    //add code above this line

    screen.exitonclick();
    return 0;

}
```

The base case for this function is a bit different. In previous examples, if the base case is true a value was returned. The function RecursiveTree() does not return a value, it draws on the screen instead. So the base case will be to keep recursing as long as branch_length is greater than some value. Define the base case as branch_length as being greater than 5.

```
void RecursiveTree(int branch_length, int angle, Turtle& t) {  
  
    //add function definitions below  
  
    if (branch_length > 5) {  
  
    }  
  
    //add function definitions above  
  
}
```

Start drawing the tree by going forward and turning right. Then call RecursiveTree() again, but reduce branch_length by 15. The code should run, but the tree will not look like a tree. It looks more like a curve made of a series of line segments decreasing in size.

```
void RecursiveTree(int branch_length, int angle, Turtle& t) {  
  
    //add function definitions below  
  
    if (branch_length > 5) {  
        t.forward(branch_length);  
        t.right(angle);  
        RecursiveTree(branch_length - 15, angle, t);  
    }  
  
    //add function definitions above  
  
}
```

In main(), let's call the RecursiveTree() function and pass in some initial values.

```

int main(int argc, char** argv) {

    //add code below this line

    TurtleScreen screen(400, 300);
    Turtle tina(screen);
    RecursiveTree(45, 20, tina);

    //add code above this line

    screen.exitonclick();
    return 0;

}

```

The next step is to draw the branch that goes off to the left. Since the turtle turned to the right the number of degrees that the parameter angle represents, the turtle needs to turn to the left twice the degrees of angle. Turning to the left angle will put the turtle back at its original heading. The turtle needs to go further to the left. Then draw another branch whose length is reduced by 15.

```

void RecursiveTree(int branch_length, int angle, Turtle& t) {

    //add function definitions below

    if (branch_length > 5) {
        t.forward(branch_length);
        t.right(angle);
        RecursiveTree(branch_length - 15, angle, t);
        t.left(angle * 2);
        RecursiveTree(branch_length - 15, angle, t);
    }

    //add function definitions above

}

```

The tree is looking better, but there are two more things that need to be done. First, put the turtle back to its original heading by turning right angle degrees. Then go backwards the length of the branch. If you tweak some of the arguments when calling the RecursiveTree() function, you might notice the tree changing.

```

void RecursiveTree(int branch_length, int angle, Turtle& t) {

    //add function definitions below

    if (branch_length > 5) {
        t.forward(branch_length);
        t.right(angle);
        RecursiveTree(branch_length - 15, angle, t);
        t.left(angle * 2);
        RecursiveTree(branch_length - 15, angle, t);
        t.right(angle);
        t.backward(branch_length);
    }

    //add function definitions above

}

```

challenge

What happens if you:

- Increase the branch length argument when calling RecursiveTree() in main() for the first time such as 35 or 60?
- Increase and decrease the angle argument when calling RecursiveTree() in main() for the first time such as 10 or 40?
- Change all of the recursive cases in the function from branch_length - 15 to something smaller like branch_length - 5?
- Change the base case to if (branchLength > 1) in the function?
- Rotate the tina 90 degrees to the left before calling RecursiveTree() in main() for the first time?

▼ Code Sample

```

////////// DO NOT EDIT HEADER! //////////
#include <iostream>                //
#include "CTurtle.hpp"            //
#include "CImg.h"                 //
using namespace cturtle;         //
using namespace std;             //
////////////////////////////////////////

/**
 * @param branch_length An integer

```

```

* @param angle The angle of degree
* @param t A Turtle object
* @return A drawing symbolizing a tree branch
*/
void RecursiveTree(int branch_length, int angle, Turtle& t) {

    //add function definitions below

    if (branch_length > 5) {
        t.forward(branch_length);
        t.right(angle);
        RecursiveTree(branch_length - 5, angle, t);
        t.left(angle * 2);
        RecursiveTree(branch_length - 5, angle, t);
        t.right(angle);
        t.backward(branch_length);
    }

    //add function definitions above
}

int main(int argc, char** argv) {

    //add code below this line

    TurtleScreen screen(400, 300);
    Turtle tina(screen);
    tina.left(90); //rotates Turtle's original position
    tina.speed(TS_FASTEST); //speeds up Turtle's movement
    RecursiveTree(35, 20, tina);

    //add code above this line

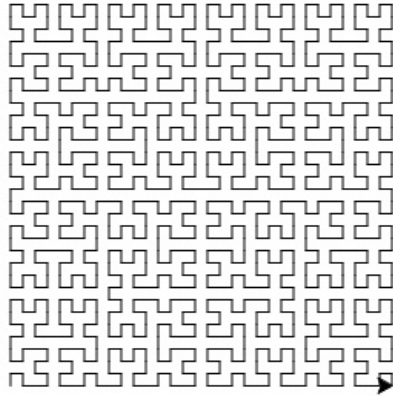
    screen.exitonclick();
    return 0;

}

```

Lab 2

Lab 2 - The Hilbert Curve



Hilbert Curve

The Hilbert Curve is a fractal, space-filling curve. Start by creating a Turtle object, and then write the function header for the recursive function `Hilbert`. The parameters for the function are the distance the turtle will travel, the rule to be used, an angle (determines how tight the fractal is), depth (how intricate the fractal is), and the Turtle object.

This is the code you have so far in the text editor:


```

////////// DO NOT EDIT HEADER! //////////
#include <iostream>                //
#include "CTurtle.hpp"            //
#include "CImg.h"                  //
using namespace cturtle;         //
using namespace std;             //
////////////////////////////////////////

/**
 * @param dist, integer
 * @param rule, integer
 * @param angle, integer
 * @param depth, integer
 * @param t, Turtle
 * @return A drawing of the Hilbert Curve
 */
void Hilbert(int dist, int rule, int angle, int depth, Turtle&
t) {

    //add function definitions below

    //add function definitions above

}

int main(int argc, char** argv) {

    //add code below this line

    TurtleScreen screen(400, 300);
    Turtle tina(screen);

    //add code above this line

    screen.exitonclick();
    return 0;

}

```

The base case for the function is when depth is 0. Another way to think about the base case is that if depth is greater than 0, keep drawing the fractal. Use `if (depth > 0)` as the base case. Also, there are two rules for the turtle. Ask if rule is equal to 1 or if it is equal to 2.

```

void Hilbert(int dist, int rule, int angle, int depth, Turtle&
            t) {

    //add function definitions below

    if (depth > 0) {
        if (rule == 1) {
            //rule1 code
        }
        if (rule == 2) {
            //rule2 code
        }
    }

    //add function definitions above

}

```

The code continues with if rule is equal to 1, then the turtle is going to turn left, recursively call the Hilbert() function with rule set to 2, go forward, turn right, recursively call the Hilbert() function with rule set to 1, go forward, recursively call the Hilbert() function with rule set to 1, turn right, and finally move forward. Because the base case is based on depth, it must be reduced by 1 each time the Hilbert() function is called recursively.

```

if (rule == 1) {
    //rule1 code
    t.left(angle);
    Hilbert(dist, 2, angle, depth - 1, t);
    t.forward(dist);
    t.right(angle);
    Hilbert(dist, 1, angle, depth - 1, t);
    t.forward(dist);
    Hilbert(dist, 1, angle, depth - 1, t);
    t.right(angle);
    t.forward(dist);
    Hilbert(dist, 2, angle, depth - 1, t);
    t.left(angle);
}

```

If rule is equal to 2, then the code is almost the inverse of when rule is equal to 1. The turtle will still go forward, but left turns become right turns, right turns become left turns, and recursive calls to Hilbert() will use 2 instead of 1 for the rule parameter (and vice versa).

```

if (rule == 2) {
    //rule2 code
    t.right(angle);
    Hilbert(dist, 1, angle, depth - 1, t);
    t.forward(dist);
    t.left(angle);
    Hilbert(dist, 2, angle, depth - 1, t);
    t.forward(dist);
    Hilbert(dist, 2, angle, depth - 1, t);
    t.left(angle);
    t.forward(dist);
    Hilbert(dist, 1, angle, depth - 1, t);
    t.right(angle);
}

```

Finally, call the `Hilbert()` function in `main()` and run the program to see the fractal.

```

int main(int argc, char** argv) {

    //add code below this line

    TurtleScreen screen(400, 300);
    Turtle tina(screen);
    Hilbert(5, 1, 90, 5, tina);

    //add code above this line

    screen.exitonclick();
    return 0;

}

```

▼ Speeding up the turtle

The Hilbert Curve can be slow to draw. You can change the speed of the Turtle `tina` with the following command `tina.speed(TS_FASTEST)`; before calling the `Hilbert()` function.

challenge

What happens if you:

- Change the dist parameter to 3?
- Start with the rule parameter as 2?
- Change the angle parameter to 85?
- Change the depth parameter to 4?

Note: You might need to adjust your TurtleScreen's size to capture all of the output.

▼ Sample Code

```
////////// DO NOT EDIT HEADER! //////////  
  
#include <iostream> //  
#include "CTurtle.hpp" //  
#include "CImg.h" //  
using namespace cturtle; //  
using namespace std; //  
////////////////////////////////////////  
  
/**  
 * @param dist, integer  
 * @param rule, integer  
 * @param depth, integer  
 * @param t, Turtle  
 * @return A drawing of the Hilbert Curve  
 */  
void Hilbert(int dist, int rule, int angle, int depth, Turtle&  
             t) {  
  
    //add function definitions below  
  
    if (depth > 0) {  
        if (rule == 1) {  
            //rule1 code  
            t.left(angle);  
            Hilbert(dist, 2, angle, depth - 1, t);  
            t.forward(dist);  
            t.right(angle);  
            Hilbert(dist, 1, angle, depth - 1, t);  
            t.forward(dist);  
            Hilbert(dist, 1, angle, depth - 1, t);  
            t.right(angle);  
            t.forward(dist);  
        }  
    }  
}
```

```

        Hilbert(dist, 2, angle, depth - 1, t);
        t.left(angle);
    }
    if (rule == 2) {
        //rule2 code
        t.right(angle);
        Hilbert(dist, 1, angle, depth - 1, t);
        t.forward(dist);
        t.left(angle);
        Hilbert(dist, 2, angle, depth - 1, t);
        t.forward(dist);
        Hilbert(dist, 2, angle, depth - 1, t);
        t.left(angle);
        t.forward(dist);
        Hilbert(dist, 1, angle, depth - 1, t);
        t.right(angle);
    }
}

//add function definitions below

}

int main(int argc, char** argv) {

    //add code below this line

    TurtleScreen screen(400, 300);
    Turtle tina(screen);
    tina.speed(TS_FASTEST);
    Hilbert(8, 1, 90, 4, tina);

    //add code above this line

    screen.exitonclick();
    return 0;

}

```

Lab Challenge

Lab Challenge

Problem

Write a recursive function called `RecursivePower()` that takes **two integers** as parameters. The first parameter is the base number and the second parameter is the exponent. Return the base number parameter to the power of the exponent.

DO NOT edit any existing code or you will not receive credit for your work!

```
#include <iostream>
using namespace std;

//add function definitions below this line

//add function definitions above this line

int main(int argc, char** argv) {
    cout << RecursivePower(stoi(argv[1]), stoi(argv[2])) << endl;
    return 0;
}
```

Expected Output

* If the function call is `RecursivePower(5, 3)`, then the function will return 125.

* If the function call is `RecursivePower(4, 5)`, then the function will return 1024.

Compile and test your code with a few different values

▼ Expected Output

216

▼ Expected Output

1