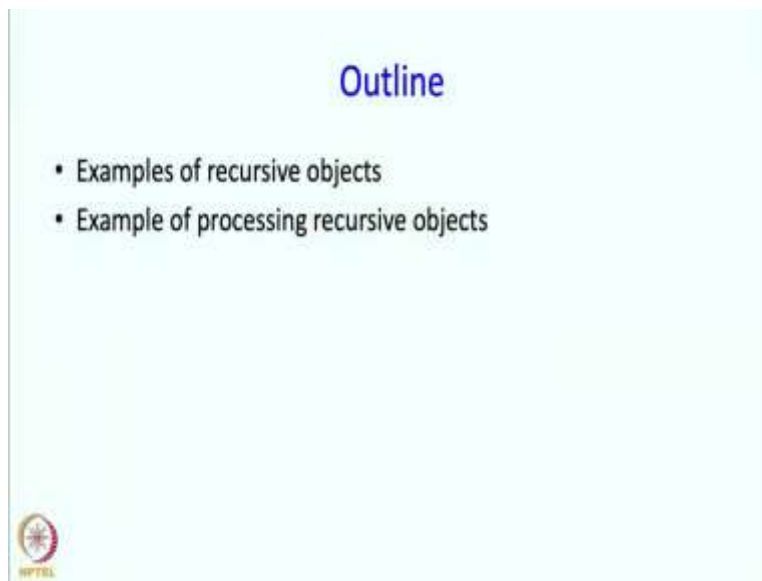


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 11 Part - 2
Recursion
Recursive objects, Tree Drawing

(Refer Slide Time: 0:36)



Welcome back, in the previous segment we defined recursion and we saw an example of it. In this segment we are going to look at recursive objects. So I am going to give you some examples of recursive objects and then also an example of processing recursive objects using a recursive function.

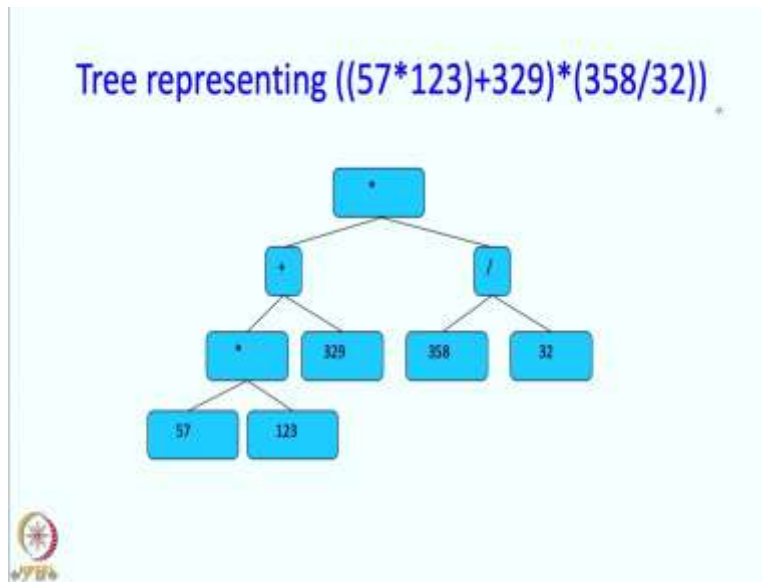
(Refer Slide Time: 0:40)



So, here is a picture of what is often called an organization tree. It is called a tree because you can think of this as the root and then these are the branches. And so here, the tree sort of is growing downwards, but that is ok. So, the important point is that there is something that this is sort of the, the original the root or the original trunk or whatever and then there are these branches coming out coming out of it.

Now, an organization which is represented by this tree, can somehow be thought of as a recursive object. Why? Because this part or this division of that organization is itself a mini organization and in this sense it confirms to what we said earlier that a part of the object is like the object itself. And even here, this is kind of a mini organization.

(Refer Slide Time: 1:50)

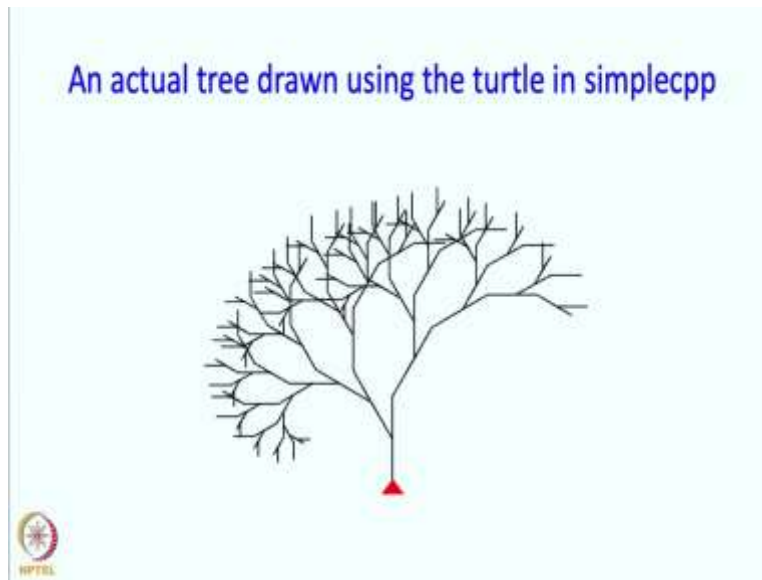


Here is another example. We have an arithmetic expression written down over here, ok. So, this expression can also be represented by this tree like diagram, so let me tell you what this diagram means. So over here, these boxes are representing just the numbers 57 and 123. So, these two numbers. Now these numbers feed into this box, which is a multiplier box. So, this box takes the numbers that it receives computes their products and sends them off to the output that it has upwards over here.

This box receives 329 from this box and the product over here. So, this is exactly like this plus, this plus and plus are really doing the same duty. So, this plus says that add up whatever values are being received and send them off to the top star, which is the star over here, and this star also receives values from the side. So, these values are 358 divided by 32. So, this operator is represented by this box and the quotient will be sent off to the side and finally the product will be sent off from this route.

So, this expression is also a tree like but it also has parts which are similar to itself. So for example, this portion is another small expression or a sub expression. So again, the entire object, which is this big expression is made up of parts, which are of the same kind, so, which are also expressions.

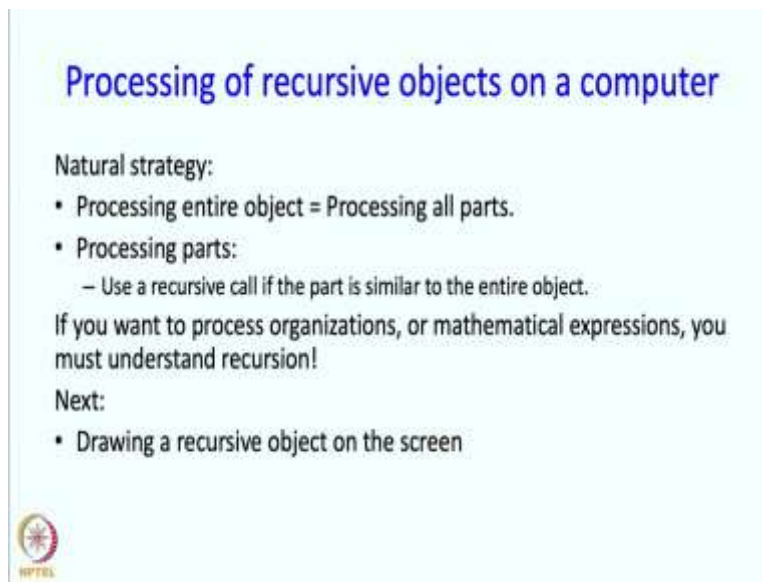
(Refer Slide Time: 3:42)



Here is a physical object which is which also has this property. So, this is a tree and this has been drawn by using the turtle. So therefore, you see this triangle the turtle over here but, never mind, do not worry about that right now. So, what does, so, what are the parts of this tree? So, first of all, there is this part, which is the trunk, ok? But, then if you look at this side, this portion is itself a tree. Ok, it is a small tree, but it really can be thought of as a tree. And on this side, there is another portion. And in this case, this portion really happens to be identical to this, this is sort of just a rotated version of this. But in any case, this is another portion, which is a part of this big tree, but which is also like a tree.

So indeed, this big tree is an object which has parts which are smaller trees. And in that sense, this big tree is a recursive object. Just like expressions were recursive objects and organizations were also recursive objects. Now, we will have occasion to process recursive objects on a computer. In fact, we want to process everything. So certainly recursive objects. And the natural strategy to process recursive objects turns out to be the following.

(Refer Slide Time: 5:12)



Processing of recursive objects on a computer


Natural strategy:

- Processing entire object = Processing all parts.
- Processing parts:
 - Use a recursive call if the part is similar to the entire object.

If you want to process organizations, or mathematical expressions, you must understand recursion!

Next:

- Drawing a recursive object on the screen

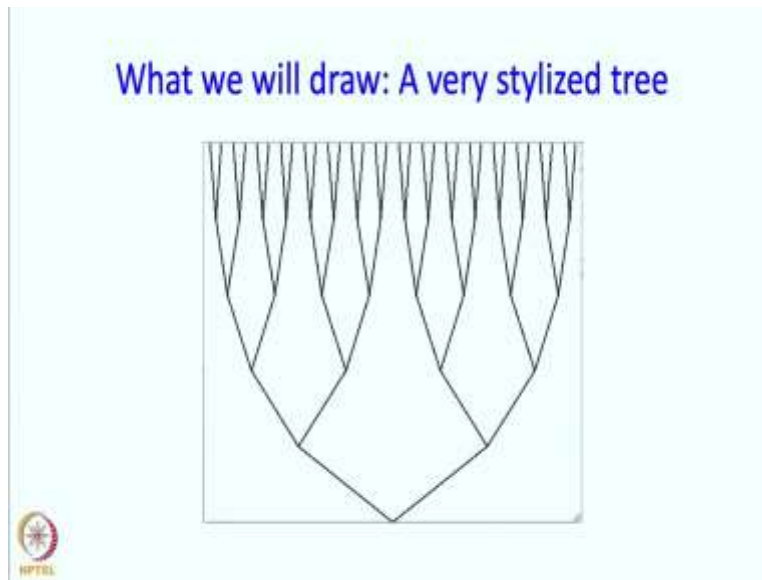


So the processing, the entire object is the same as processing all parts. And how do we process the parts? So, we are going to use a recursive call if the part is similar to the entire object. So, if it is just a smaller version of the entire object, we do not need to write new code, but we can use the code that we were writing for the bigger object itself, ok. And in fact, this is exactly what happened when we wrote the code for GCD.

So, if you want to process organizations, or mathematical expressions, you really must understand recursion. And recursion indeed is sort of one of the cornerstones of computer science. The other, the other cornerstone is iteration in some sense, and iteration and recursion together could be said to be sort of 2 really major ideas in computer programming and also in computer science.

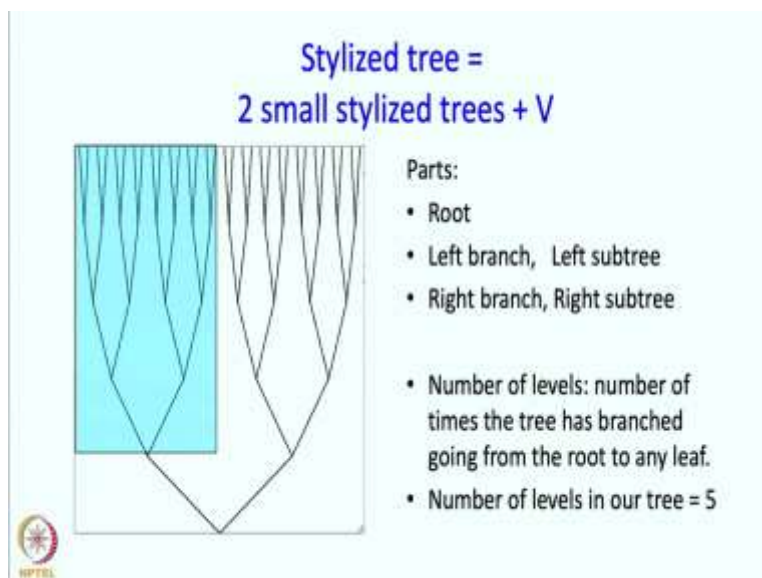
So, what we are going to do next is we are going to draw a recursive object on the screen. So, this is going to be an example of processing recursive objects. So, it is kind of a nice fun example, but it will contain all the elements, the major element that we want, that when we want to process the parts, we just have to make a recursive call. So, what are we going to draw?

(Refer Slide Time: 7:00)



So, here is a picture and this is a picture of a very stylized tree, you can think of it as a tree, that perhaps somebody who does modern art might draw. It is actually very pretty. And you might think that maybe, maybe there are some, actually there are some bushes that grow in Africa, which look like this. But anyway, this is what we want drawn. And you can see that this does have recursive structure, ok.

(Refer Slide Time: 7:34)



So, this big tree is really made up of two small trees, and I am calling this stylized tree because maybe some of you might think that oh, this does not look like a tree. So let us call it a stylized

tree. So this part, this covered with this blue rectangle is a small stylized tree. And on this side, we have another small stylized tree, and together they make up a big stylized tree, but of course we have to have this bottom portion, which is which I am calling it calling a V over here. So there is a V and then there are two small stylized trees which make up this big stylized tree.

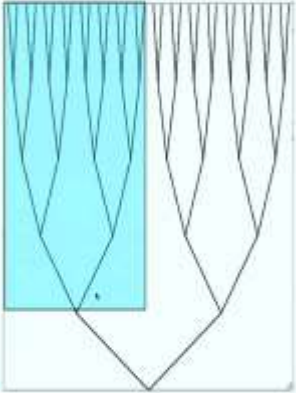
What are its parts? Well there is the root. So by root, I mean this point over here. So, we are not really going to be drawing points, but you might as well note that that point is sort of present. Then there is the left branch and then there is the left subtree. So this is the left branch and this top thing over here over here is the left subtree. And similarly, there is a right branch and then there is the right subtree.

Now, we need to have a definition, which is that of the number of levels. So we define the number of levels to be the number of times the tree has branched going from the root to the top. So the top is also often called leaf in a metaphorical sense, I mean the bottom is root and the top is a leaf. So this tree, how many times has it branched?

Well it has branched over here, it has branched over here, it has branched at this level, it has branched at this level, it has branched at this level. So, there are 5 levels at which it has branched or this tree, we will say has 5 levels.

(Refer Slide Time: 9:32)

Drawing the tree using coordinate based graphics



```
To draw an L level tree:
if L > 0{
  Draw the left branch,
  Draw a Level L-1 tree on top of it.
  Draw the right branch,
  Draw a Level L-1 tree on top of it.
}
```

- We need coordinates ...
 - Say root is to be drawn at (rx, ry)
 - Total height of drawing is H .
 - Total width of drawing is W .
- We should then figure out where the roots of the subtrees will be, and their width and height.

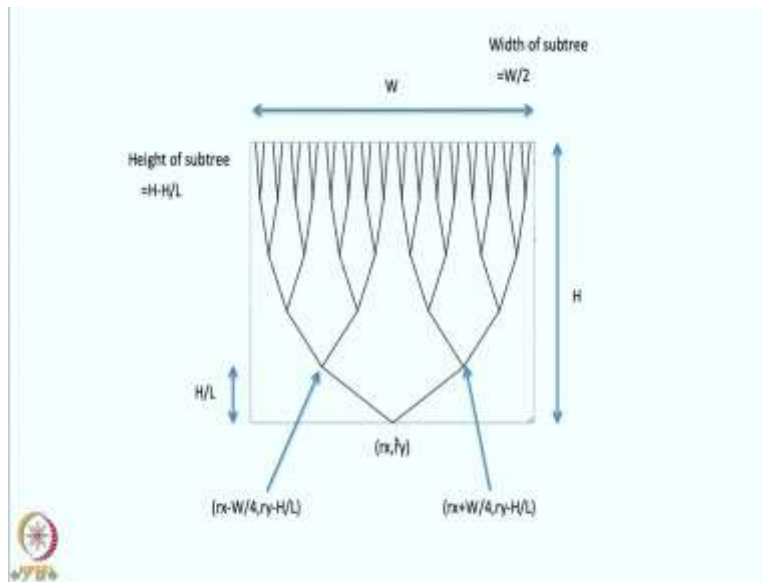
Alright, so we can now think about drawing the tree so the tree could be drawn using turtle, but we are going to try it using coordinate based graphics. So, how are we going to try it? Well, so if we are going to draw an L level tree, ok if L is equal to 0, so it is just the route, then we are really not going to do anything, I mean we could put down the point, but we will just be sloppy and not even put down a point.

So, if L is greater than 0 then we have to do something. So, what is it that we have to do? So, well we are going to draw this left branch, ok then we are going to draw this tree on top of it, but this time the number of levels in it has reduced because we have used up one level of branching over here. So, we are going to draw a level L minus 1 tree on top of it. Then we are going to draw the right branch which is this branch and then draw an L minus 1 level tree on top of it. So, that is this tree portion. So, that is basically it. But of course, in order to draw we need the coordinates. And so when he say draw, we need to specify the coordinates, we need to tell our program where we want the tree drawn. So maybe we will one of the arguments could be RX and RY which define the point at which the root is supposed to be drawn. So, the root we are going to tell our function that it has to be drawn at RX , RY , ok. Then we also should tell our function what is the total height of the drawing. So by height I mean this entire the extent the this extent. So, let us say we tell that and we use the name H for it. Similarly, we tell that the width is W .

Now, we have told our function that the root must be present over here and the width must be W and the height must be H . So, we have done our job in the sense of specifying the region in which the tree should be drawn. Well, we will make some ad hoc decisions say that the levels in this picture are equally spaced. So once we say that we have we have done our job.

Now, the recursive call must figure out where the smaller trees should be placed. So, when it when the recursive call is made, it should be made with the proper height and width and also the proper route positions. So, we need some calculation of these coordinates and we are going to see this next.

(Refer Slide Time: 12:36)



So this is what we said was the requirement for drawing the big tree. So we are given R_X , R_Y . We are given W and we are given H . What we want to know? We want to know the position of this root because then we could, we could issue the command for the smaller tree to be drawn and also we could issue the command for drawing this branch. Similarly, we need the position of this root and we need the height and width of the smaller trees, ok. So, let us try to work it out.

So this distance, this is going to be the y coordinate this level is the same as the level of the y coordinate of the roots, ok. So we said that the levels are going to be equally spaced, so the entire distance is H and therefore, this height is going to be H upon L . Then this is the route and its x position is going to be half day of this, ok. So, what is it going to be?

So this distance from here to here is R_X , let us say, ok, and this entire distance is half of the W and therefore, this distance is W by 4. And the y coordinate of this point is R_Y minus H upon L , why minus? Because if you remember the coordinates y coordinate goes downwards. So we are going back a distance (H/L) H by L . So, we have figured out what the coordinates of this position are going to be.

So now we can instruct our recursive call as to where its root has to be drawn. What are going to be the coordinates of this? So well, that is easy. So, instead of going back W by 4, we have to go forward W by 4 and therefore, the x coordinate is R_X plus W by 4, the y coordinate is the same

R_Y minus H by L . So all that remains now is to say how much the height and the width are going to be of the small trees.

So, the height of the sub tree is going to be this entire portion so going from here to here. So, we just have to subtract H by L from it, ok? So, it is going to be H minus H by L and the width is going to be just half the width. So, this is going to be W by 2 . So, we are more or less done, ok. So, we just have to write the program now, but we know all the coordinates. So, one more point we need to draw the branches. So the branches should go from this point to this point. So we know that those coordinates as well. So basically, we have we have done that we have we are done with the calculations.

(Refer Slide Time: 15:38)

```
void tree(int L, double rx, double ry,
          double H, double W){
    // L levels, Root at (rx,ry), Height H, Width W
    if(L>0){
        Line left(rx, ry, rx-W/4, ry-H/L);
        Line right(rx, ry, rx+W/4, ry-H/L);
        right.imprint();
        left.imprint();
        tree(L-1, rx-W/4, ry-H/L, H-H/L, W/2);
        tree(L-1, rx+W/4, ry-H/L, H-H/L, W/2);
    }
}
main_program{
    initCanvas(); tree(5, 250, 300, 300,500);
}
```

Ok, so here is what the program looks like. We have, we will call this function tree. It is not going to return anything, but it is just going to draw, ok. And its first, the first argument to it or the first parameter of this function is going to be L , the number of levels. It is going to have as parameters R_X , and R_Y which are the positions of the roots and the height and width of the region in which the tree has to be drawn. So, let me just write down sort of the main, the, the, some explanation about what these arguments are, L is the number of levels, the root is at R_X , R_Y and the height is H and the width is W , ok?

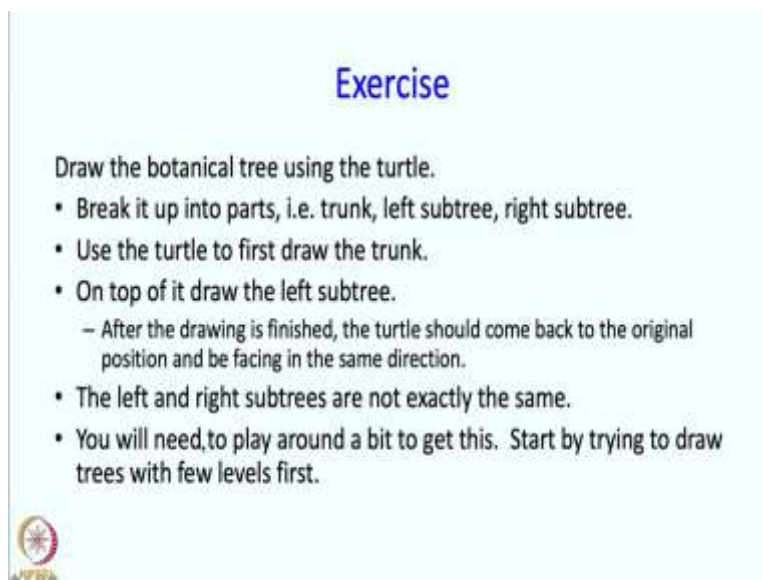
Now, this code is pretty much like what we had written in the slide earlier, if L is equal to 0 , nothing is to be done. If L is greater than 0 , then we are going to draw a line which represents the

left branch, and it is going to start at the root and it is going to go to the root of the sub tree. So these are the positions of the x and y coordinates of the root of the sub tree the left sub tree, then the right branches are we are going to start at the root of the main tree. And it is going to go to the root of the left the right sub tree. And as you if you remember from last, the last slide, these are exactly those coordinates.

Now, these left and right, the left and right branches are created locally inside, inside this inside this call, ok? And they are going to go away once we exit. So, if we want the tree picture to survive, we have to imprint them. So that is what we are doing. We are doing over here, we are imprinting the left and right branches.

And then we are going to draw the sub trees, so this, the left sub tree is being drawn over here. It is going to have L-1 levels. These are going to be the coordinates of where the route is going to be located. This is going to be the height of the sub tree and this is going to be the width of the sub tree. Similarly, we will draw the right sub tree. The only difference is that the x coordinate will be further to the right, that is it. And I can have a main program which does this. And in fact, this will end up drawing the entire thing.


(Refer Slide Time: 18:28)



Exercise

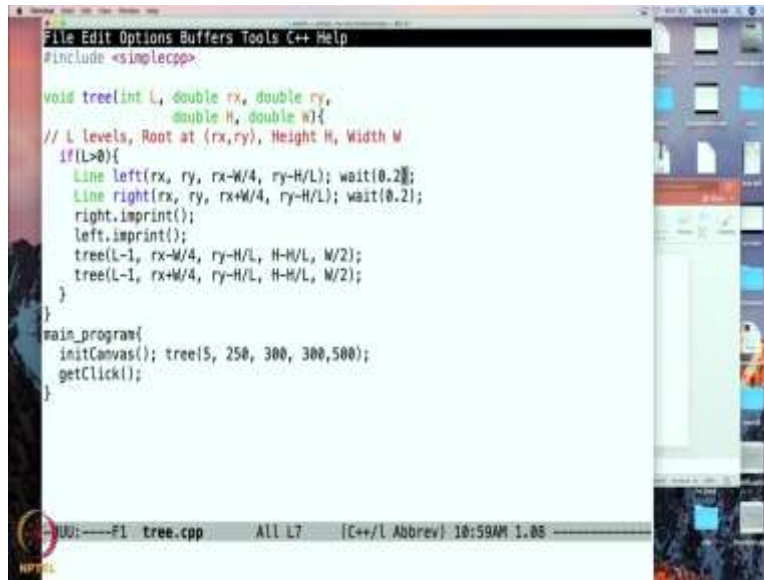
Draw the botanical tree using the turtle.

- Break it up into parts, i.e. trunk, left subtree, right subtree.
- Use the turtle to first draw the trunk.
- On top of it draw the left subtree.
 - After the drawing is finished, the turtle should come back to the original position and be facing in the same direction.
- The left and right subtrees are not exactly the same.
- You will need, to play around a bit to get this. Start by trying to draw trees with few levels first.



Ok, so at this point, let me stop and this is the exercise that you are supposed to do. But let me stop for a second over here and let us actually draw this tree. So now we are going to see a demo of this. So, let me get out of the presentation.

(Refer Slide Time: 18:52)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

void treeint L, double rx, double ry,
           double H, double W){
    // L levels, Root at (rx,ry), Height H, Width W
    if(L>0){
        Line left(rx, ry, rx-W/4, ry-H/L); wait(0.2);
        Line right(rx, ry, rx+W/4, ry-H/L); wait(0.2);
        right.imprint();
        left.imprint();
        tree(L-1, rx-W/4, ry-H/L, H-H/L, W/2);
        tree(L-1, rx+W/4, ry-H/L, H-H/L, W/2);
    }
}

main_program{
    initCanvas(); tree(5, 250, 300, 300,500);
    getClick();
}
```

And here, I have the program typed in, there are a few small changes. So, the program is the same except that after during each line I am putting a wait statement, because if I do not put these wait statements, then the drawing will happen very fast. And you will not see the order in which things are getting drawn. So, it is the order in which things are getting drawn is actually quite interesting. And finally, I also put in a get click statement, so that the program does not execute and remove the picture immediately.

(Refer Slide Time: 19:26)



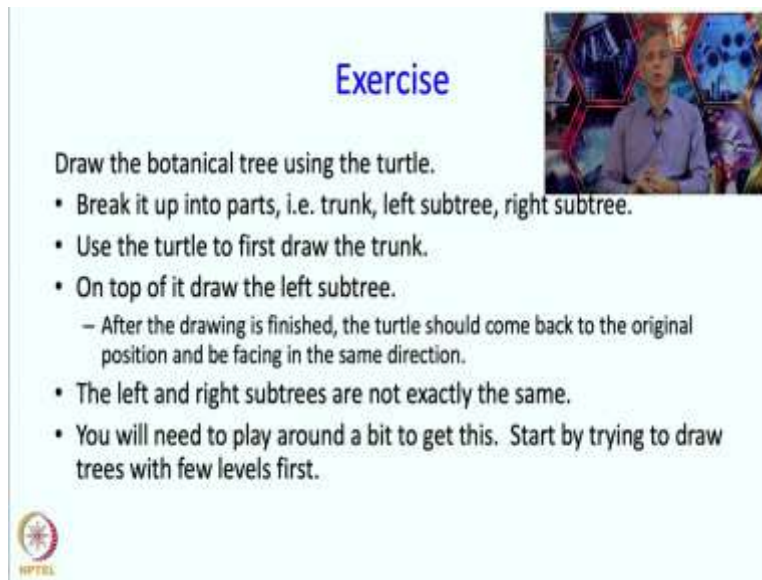
```
~/Desktop/npTEL/week5 : s++ tree.cpp
+ g++ tree.cpp -Wall /Users/abhiram/simplecpp/lib/libspri.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week5 : ./a.out
~/Desktop/npTEL/week5 : %
emacs -nw recursiveGcd.cpp

[1]+ Stopped                  emacs -nw recursiveGcd.cpp
~/Desktop/npTEL/week5 : s++ tree.cpp
+ g++ tree.cpp -Wall /Users/abhiram/simplecpp/lib/libspri.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week5 : ./a.out
~/Desktop/npTEL/week5 : %
emacs -nw recursiveGcd.cpp

[1]+ Stopped                  emacs -nw recursiveGcd.cpp
~/Desktop/npTEL/week5 : %
emacs -nw recursiveGcd.cpp

[1]+ Stopped                  emacs -nw recursiveGcd.cpp
~/Desktop/npTEL/week5 : s++ tree.cpp
+ g++ tree.cpp -Wall /Users/abhiram/simplecpp/lib/libspri.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week5 : s++ tree.cpp
+ g++ tree.cpp -Wall /Users/abhiram/simplecpp/lib/libspri.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week5 : ./a.out
```



(Refer Slide Time: 20:39)



Exercise

Draw the botanical tree using the turtle.

- Break it up into parts, i.e. trunk, left subtree, right subtree.
- Use the turtle to first draw the trunk.
- On top of it draw the left subtree.
 - After the drawing is finished, the turtle should come back to the original position and be facing in the same direction.
- The left and right subtrees are not exactly the same.
- You will need to play around a bit to get this. Start by trying to draw trees with few levels first.



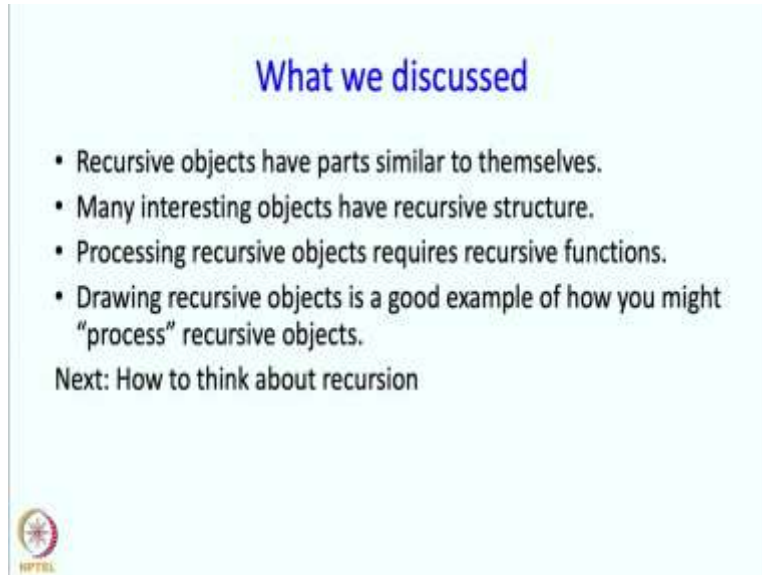
So, then I am going to leave you with an exercise. So the exercise is also there in the book and some more hints are given there. So you should certainly look at that. But, you are supposed to draw the real tree, the botanical tree that I showed you earlier, and some ideas on how to draw it? Well, break it up into parts. So say there might be the trunk, there might be the left sub tree and right sub tree.

And this time it turns out it is convenient to try it using the turtle rather than using these coordinate based graphics, so what you should do is, you should get turtle to first draw the tree and on top of it, draw the left sub tree or somewhere attached to that. Ok, after the drawing is finished, and this is the crucial part, you want the turtle to come back to the original position because only then you can draw the next because if the turtle did not come back to the original position, that is not good because you want it to be at a proper position. So that only then you know where to move it, so as to get to the right position.

And then, if you observe that the structure of the tree you will see that the left and right sub trees are not exactly the same. So, the right sub tree is, first of all, it does not originate at the same place and second it is rotated a little bit, ok. So, you will have to play around, you will have to play around a little bit to get this. And a suggestion is that do not try to draw the whole thing at once. But first draw a tree with small number of levels. So maybe just start with level equal to 1

and just see whether your program is trying just the trunk that should be easy, but then move on to level 1, level 2 and so on.


(Refer Slide Time: 22:38)



What we discussed

- Recursive objects have parts similar to themselves.
- Many interesting objects have recursive structure.
- Processing recursive objects requires recursive functions.
- Drawing recursive objects is a good example of how you might “process” recursive objects.

Next: How to think about recursion



Ok, so what did we discuss in this segment? So, we said that recursive objects have parts similar to themselves. And we saw examples of many interesting objects which have this recursive structure. Then we said that processing recursive objects requires recursive functions. And drawing recursive objects is a good example of how you might process recursive objects.

But, later in this course and certainly in other courses, if you care to take them, there will be there will be much more substantial processing of recursive objects that you will have occasion to do. So, this is the end of the current segment. In the next segment, we are going to say a little bit more about how we should be thinking about recursion. So, let us take a break.