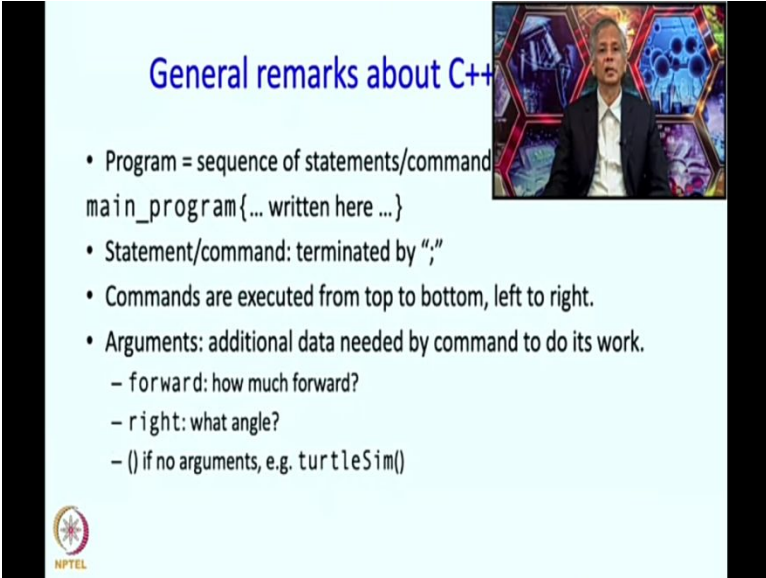


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 1 Part – 3
Introduction
Program Structure and Syntax

Welcome back. In the previous segment, we discussed the following things: We discussed the repeat statement, and the cin, cout statements and some commands to compute mathematical functions. Actually, we also discussed the statement by which we can reserve locations or cells in memory.

(Refer Slide Time: 0:46)




General remarks about C++

- Program = sequence of statements/command

```
main_program{... written here ...}
```

- Statement/command: terminated by “;”
- Commands are executed from top to bottom, left to right.
- Arguments: additional data needed by command to do its work.
 - forward: how much forward?
 - right: what angle?
 - () if no arguments, e.g. turtleSim()



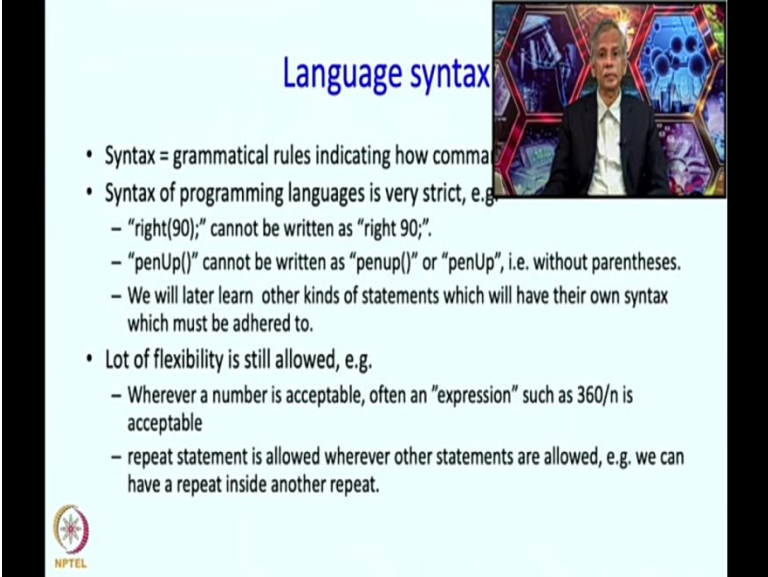
Now, in this segment, I am going to make some general remarks about C++ programs. First of all, let me observe that a program is really a sequence of statements or commands, and it has to be written in a particular manner. So the keyword ‘main_program’ must appear first, followed by an open brace. After that, the sequence of statements representing the main program have to be put in, and finally, the closing brace has to be put in.

A statement or a command is typically terminated by a semicolon. So the semicolon is the C++ equivalent of the full stop of the English language. Just as in the English language, a sentence

ends with a full stop, in C++ a statement or a command ends with a semicolon. And just as you read sentences top to bottom, left to right on a page written in the English language, similarly, C++ programs are also executed by reading the statements top to bottom and left to right.

Now, some of the statements require additional data for them to do their work. For example, if I write 'forward' I need to tell the computer how much I want to move forward, so such data is called an 'argument'. Similarly, 'right' needs to be told how much angle. So this is another argument, there can be commands which require more than one argument, and we will see them soon. And there can be commands which do not require any argument at all. So, 'turtleSim()' for example, did not require any argument, and so in that case, we just put that opening and closing parenthesis immediately.

(Refer Slide Time: 2:53)



The slide is titled "Language syntax" in blue text. It features a list of bullet points on the left and a small video inset on the right showing a man in a suit speaking. The video inset is framed by a colorful, abstract graphic. In the bottom left corner of the slide, there is a logo for "HPTTEL" with a star-like symbol.

- Syntax = grammatical rules indicating how commands must be written
- Syntax of programming languages is very strict, e.g.
 - "right(90);" cannot be written as "right 90;"
 - "penUp()" cannot be written as "penup()" or "penUp", i.e. without parentheses.
 - We will later learn other kinds of statements which will have their own syntax which must be adhered to.
- Lot of flexibility is still allowed, e.g.
 - Wherever a number is acceptable, often an "expression" such as 360/n is acceptable
 - repeat statement is allowed wherever other statements are allowed, e.g. we can have a repeat inside another repeat.

Often in the discussion about languages or especially programming languages the term 'syntax' is used. The term syntax is used to refer to the form or the grammar which the language is supposed to follow. So syntax refers to the grammatical rules indicating how commands must be written and the syntax of programming languages is very strict. What do I mean by that?

Well, 'right(90)' cannot be written in any other form. For example, you cannot write it as right-space-90. That is not allowed, the parentheses are necessary. However, you could put a

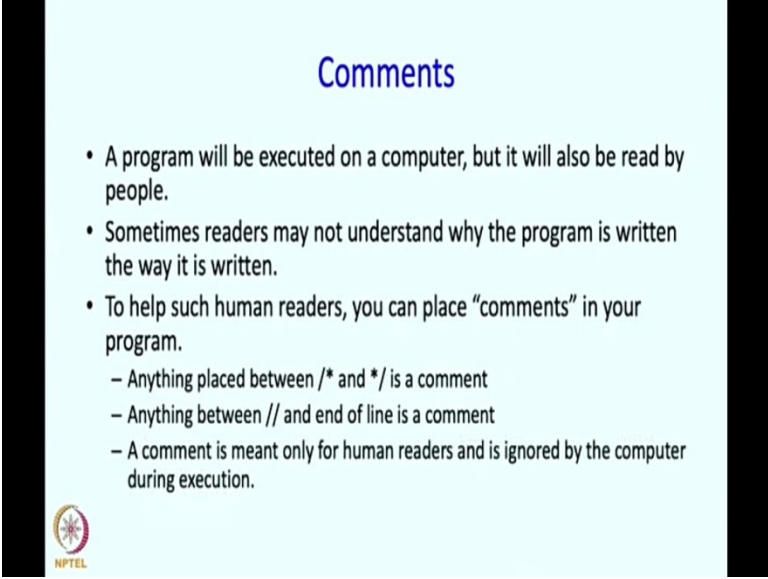
space before and after the parenthesis; that is allowed. However, you cannot split the word 'right', so you cannot write r-space-i-g-h-t, that is not allowed. You cannot even capitalize the 'r', okay? So it is strict in a certain sense and you have to understand in what sense the language is strict, and you have to adhere to it very very very very precisely.

So, 'penUp()', for example cannot be written as 'penup()', so the 'u' is expected to be capitalized. Similarly, the parentheses are needed, so you cannot just leave out the parentheses. There are other statements also which we will learn and they will also have their syntax and programs will have to be written strictly in that syntax. You might say that look oh what is the big difference between 'penup' and 'penUp', is not it common sense? Well it may be common sense to you and me, but a computer requires you to be exact, so a computer will think of 'penUp' as something different from 'penup'. So, if you want 'Up' you have to say capital up.

However, in spite of all this, there is a lot of flexibility allowed. So for example, I already said that you can put spaces which do not split a word or split a number, so you can put a space before or after the parentheses. In addition there are some other natural features as well. So, wherever we say that look put a number over here, usually you will allow a numerical expression also. So for example in a repeat or in a right rather in the right we can put an expression such as 360 divided by nsides, or it could be n, if n is the name of a variable.


So, wherever a number is going to appear you can put in the name of a cell in memory or you can put in an expression, which depends on some values, so that kind of flexibility is there. And also there is another flexibility which is that we talked about the repeat statement which has a body and we said that the body consists of several statements, well one of these statements could be a repeat statement itself.

(Refer Slide Time: 5: 57)



Comments

- A program will be executed on a computer, but it will also be read by people.
- Sometimes readers may not understand why the program is written the way it is written.
- To help such human readers, you can place “comments” in your program.
 - Anything placed between `/*` and `*/` is a comment
 - Anything between `//` and end of line is a comment
 - A comment is meant only for human readers and is ignored by the computer during execution.




We will come to the example of a repeat inside a repeat, but let me make a few additional remarks. So first of all I am going to tell you something called ‘comments’. Now, a program is going to be executed on the computer, but it will also be read by people. So you might write a program and your teacher might read it. If you are working, your boss or your colleagues might read the program you have written, or you might write a program and it might be so good, that you may give it to several other people, all of whom might read that program.

Now, if you write a program, it has to be understandable. So you have to you might want to say something more in addition to the statements on it. So for this purpose, you can place something called ‘comments’ in your program, okay.

How do you place comments? Well you can put comments between a ‘`/*`’ sequence of characters and a ‘`*/`’ sequence of characters. So whatever appears between these two sequences will be thought of as a comment. Alternatively, I can put ‘`//`’ and then I can put a comment after it and the comment ends when the line ends. So typically, the first form is used to define a long comment which extends over several lines, the second form is used to define a short comment which really extends over a single line or which really put at the end of a line okay.

Now a comment is only meant for the human readers, and the computer completely ignores, the computer does not care what you put in a comment.

(Refer Slide Time: 7:40)



```
/* Author: Abhiram Ranade
Program to draw polygon */
#include <simplecpp>
main_program{
  turtleSim();
  cout << "How many sides?";
  int nsides; cin >> nsides;
  repeat(nsides){
    forward(100);
    right(360.0/nsides); // Exterior angle of an n sided polygon is 360/n
  }
  wait(10);
}
```

The slide features a light blue background with a black border. The title 'A program with comments' is centered at the top in a blue font. Below the title, C++ code is displayed in a monospaced font. The code includes a multi-line block comment at the top identifying the author and the program's purpose. It then includes the <simplecpp> header and defines a main_program function. Inside this function, it calls turtleSim(), prompts the user for the number of sides, and uses a repeat loop to draw the polygon. A single-line inline comment explains the calculation for the exterior angle. The program concludes with a wait(10) call and a closing brace. An NPTEL logo is visible in the bottom left corner of the slide.

So we had our program for drawing that polygon, so I have put two comments in this, so at the top I have put a comment which tells who the author of that program is, and it tells the purpose of that program. So this is usually good practice. You should put your names on programs so that if somebody has questions, they can come and talk to you and also you might as well tell them what is that program going to do and then over here, we did something like dividing 360 by nsides.

A reader might look at this and say, “Look, why are they dividing 360 by nsides?” Now, if you want to help out those readers, you can say that the exterior angle of an nsided polygon is 360 by n. I want to draw an nsided polygon and therefore, I will turn by the angle 360 divided by n, or in this case, nsides.

So that is how comments help, they do not help the computer but they help human readers and you are extremely extremely emphatically encouraged to write comments. For one thing, you will help your friends and boss and your teacher, but you will realize that you write a program today, in a week or so you will completely forget what you have written. Maybe you look at this


program and ask yourself oh why did I write 360 upon nsides? To help yourself at least, put in a comment. You will be really really grateful and this is a required good practice if you become a professional programmer.

(Refer Slide Time: 9:22)

Indentation

```
#include <simplecpp>
main_program{
  turtleSim();
  cout << "How many sides?";
  int nsides; cin >> nsides;
  repeat(nsides){
    forward(100);
    right(360.0/nsides);
  }
  wait(10);
}
```

- You will notice that at the beginning of some lines some space is inserted.
- This space is called indentation.
- The indentation allows you to quickly see which statements constitute the body of the repeat, which statements are part of the main program.
- The general rule: if X is "inside" Y, then put two additional spaces before every line of X.
- Also note how the { and } are placed.
- Indentation is very helpful for human readers.



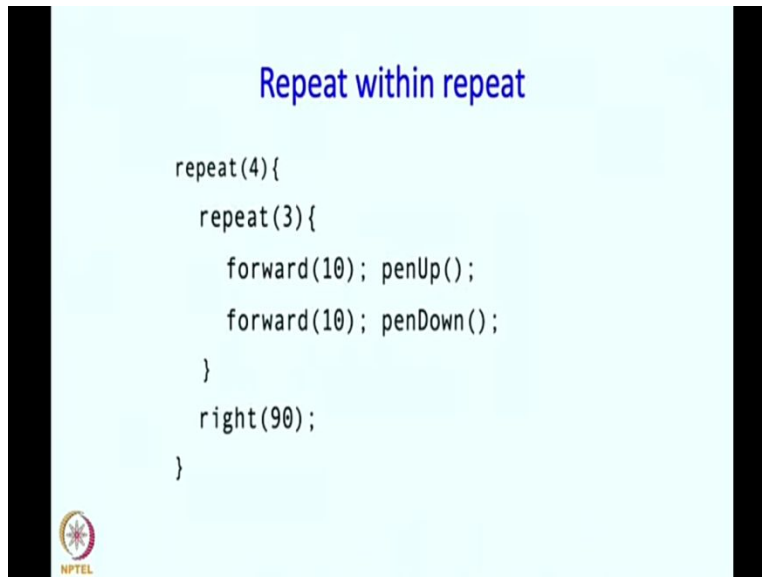
Another important practice and another part of how you write a programs is so called ‘indentation’. So here is the same program and I am going to tell you what indentation is inside it. So, notice that at the beginning of some lines some space is inserted. So for example, before this turtleSim, some space is there. In the body of the repeat there is a space which offsets it from the repeat statement itself, so this space is called indentation. The indentation allows you to quickly see which statements constitute the body of the repeat, or which statements are parts of the main program. So for example, this ‘include<simplecpp>’ is not a part of the main program, whereas all these statements are, so by offsetting, I can visually quickly know that look this is the main program over here or I can visually quickly note that oh this is not only in the main program, but it is also in a repeat statement inside the main program.

So you may guess the general rule if some x is inside some y, then typically you will put two additional spaces before every line of x. So that is what has happened over here - for the body of the repeat which is sort of supposed to be inside the repeat statement and these are the statements inside the main program so we have put two spaces over here.

Then there is also a style that you have to master, as to how you write the opening brace and the closing brace. So you can see that the closing brace reverts back to the old indentation. So again it is considered extremely unfriendly if you do not use indentation. And again, for your own

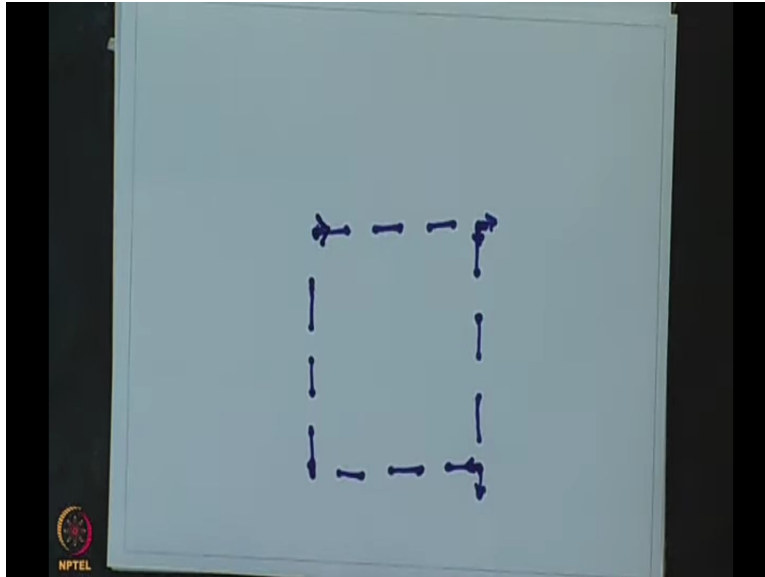
benefit, you should use indentation because when you come back to the program it will make it much easier for you to understand what you yourself have written. And of course indentation is useful for human readers, but it is ignored during execution.

(Refer Slide Time: 11:49)



Now, we said that the body of a repeat itself can contain a repeat statement. So here is an example. So we have a repeat(4) and inside that, there is a repeat(3) and a right(90) statement, so how does this execute? Well the first rule is that whatever is inside this outer repeat is going to be executed 4 times. So what is inside this outer repeat? Well there is a repeat(3) and a right(90), so we are going to execute this statement 3 times, then we are going to execute this statement once, then we are going to go back again execute thrice, execute once and so on, do this 4 times. That is what the rule is there is no real mystery as such, so let us try to figure out what this is doing.

(Refer Slide Time: 12:44)



So let us just forget one just iteration of this repeat, what is it doing? So it is going to go forward 10 steps, so let me draw that, and to begin with, let us assume that the pen is down. So here is the turtle and it goes forward 10 pixels. At this point the pen goes up, once the pen goes up the turtle goes again forward 10 steps, but it goes forward and this time the pen is not down so there is no line drawn, but at this point the pen is put down again, so this finishes one iteration, what happens in the second iteration? Again forward line is drawn again a movement is there without any line being drawn and one more iteration a line is drawn and there is a movement again okay? So this finishes one iteration of that repeat 3 statement.

After that we continue forward and now there is a right(90), so earlier the turtle was pointing in this direction, so right(90) will cause it to point in this direction. So that ends an iteration of the outer repeat. So again we start from the very beginning, so we go back to the top of the repeat(4) and we start executing, so now we are executing the second iteration. So for the second iteration again the repeat(3) is going to be executed, so this will again cause forward(10) and then penUp, and forward, but this time the pen is high so that nothing will be drawn.

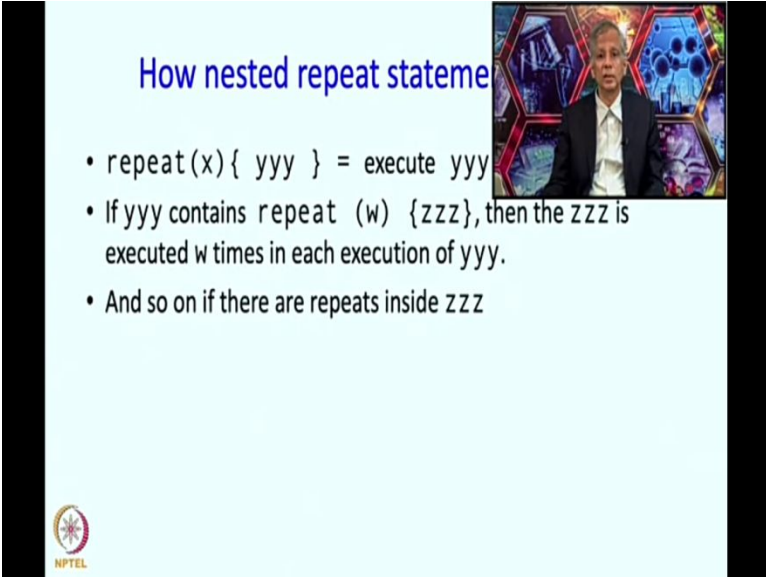
So that is just one iteration of the inner repeat, so one more iteration movement, one more iteration which consists of this movement and again we come to the bottom of the repeat(4), this

time at the end of the second iteration. So at this point the turtle was pointing in the downward direction, it now turns right and points in this direction.

So now one more iteration we will call again this movement forward so drawing but forward without movement. Movement and drawing, forward without movement. Movement and drawing, forward without movement and again we will turn right, so the turtle will start pointing in this direction, so again it will execute forward and draw, forward without draw, forward and draw, forward without draw, forward and draw, forward without draw. So the turtle will end up at the same place after that it will again turn right so it will end up facing in this direction.

So, the turtle will start facing right over here, it will draw these dashed lines, turn, draw these dashed lines, turn, draw these dashed lines, turn, draw these dashed lines and again turn and come back to the same position and have the same orientation okay. So notice that repeat within repeat is very very useful. Imagine, if I had wanted to draw a long line with lots of dashes, I just have to make that `repeat(100)`. If I wanted 100 dashes, I do not have to write 100 statements.

(Refer Slide Time: 15:47)



The slide features a light blue background with a black border. On the right side, there is a small video inset showing a man in a suit. The main text is as follows:

How nested repeat statements

- `repeat(x){ yyy }` = execute `yyy`
- If `yyy` contains `repeat (w) {zzz}`, then the `zzz` is executed `w` times in each execution of `yyy`.
- And so on if there are repeats inside `zzz`

At the bottom left, there is a circular logo with a star and the text 'NPTEL' below it.

So again, a quick summary of what I just said in that example, in general if I have a `repeat(x){yyy statement}`, it means execute 'yyy', 'x' times. Now, if 'yyy' itself contains



repeat(w){zzz}, then 'zzz' is executed 'w' times in each execution of 'yyy', or, in each iteration of that outer repeat(x), this repeat(w){zzz} is going to be executed.

And in a single execution of repeat(w){zzz}, 'zzz' is actually executed 'w' times. Now I can have as many repeats going on inside each other, or 'nested' inside each other. So what is the logic there? Exactly similar.

(Refer Slide Time: 16:32)

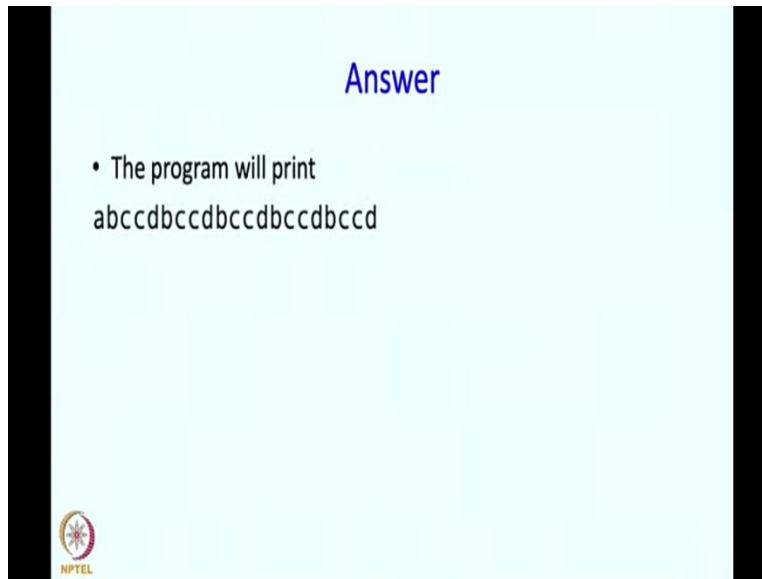
What will the following program do?

```
#include <simplecpp>
main_program{
  cout << "a";
  repeat(5){
    cout << "b";
    repeat(2){ cout << "c"; }
    cout << "d";
  }
}
```



Alright, so as a quick exercise, what do you think this program is going to do? So figure this out yourself and only then advance forward. So, give yourself a few seconds. So I will not explain it, but I will just show you the solution and so you can compare your answer with it. You can also type this program in, and you can see exactly what it does.

(Refer Slide Time: 16:58)



The slide has a light blue background with a black border. At the top center, the word "Answer" is written in blue. Below it, a bullet point states "The program will print" followed by the string "abccdbccdbccdbccdbccd". In the bottom left corner, there is a small circular logo with a star and the text "NPTEL" below it.

Answer

- The program will print
abccdbccdbccdbccdbccd

So the program will print this.

(Refer Slide Time: 17:10)



The slide has a light blue background with a black border. At the top center, the title "Some commonly used terminology" is written in blue. Below it, there are three bullet points defining terms: "Control is at statement w", "Control flow", and "Variable". Each bullet point includes a sub-point with a hyphen. In the bottom left corner, there is a small circular logo with a star and the text "NPTEL" below it.

Some commonly used terminology

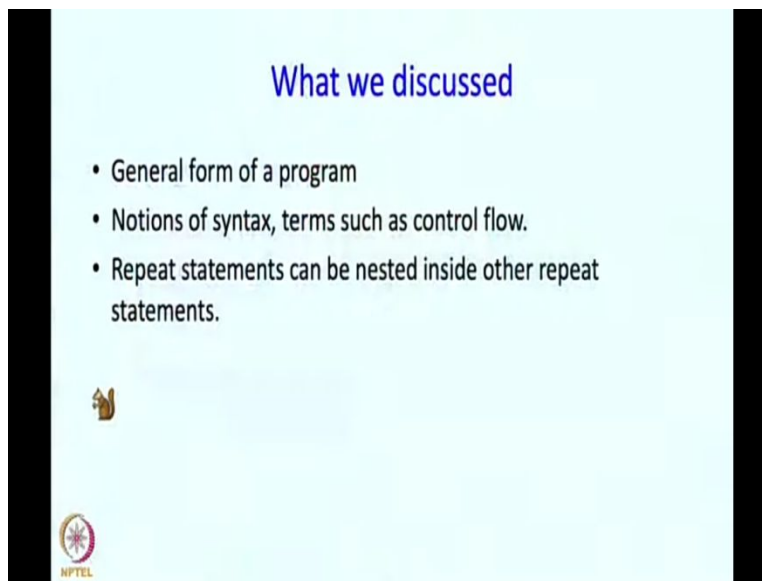
- "Control is at statement w": Computer is currently executing statement w.
- "Control flow": The order in which statements get executed.
 - Execution starts at top and goes down. Retraced if there is a repeat statement.
- Variable: region of memory designated for storing some value you need
 - Example: ns i des which we saw earlier.
 - Named so because the value stored in the region can vary

So now, I want to talk about some terms which you will hear commonly when you discuss programming. So you will hear people say that 'control' is at statement 'w', this means that the computer is currently executing statement w. Or people talk about control flow. Control flow is how the control moves or the order in which statements get executed. So execution starts at the

top and goes down and the execution is retraced if there is a repeat statement. So the control flows over and over the body of a repeat statement.

The term 'variable' is also used. A variable is simply the region of memory designated for storing some value that you need. So the spaces that we define is actually going to be called a variable. So we said that we use a cell in memory but that cell is going to be called a variable because we have asked for it, and we are going to store some specific kinds of values over there. Why is it called a variable? Well, because we are allowed to change the value that we store over there, exactly how this change happens, we will see a little bit later okay?

(Refer Slide Time: 18:32)



So here is what we discussed at this segment: we discussed the general notion of a program, we discussed the notions of syntax and terms such as control flow, and then we made this observation that repeat statements can be nested inside other repeat statements. So we will take a quick break.