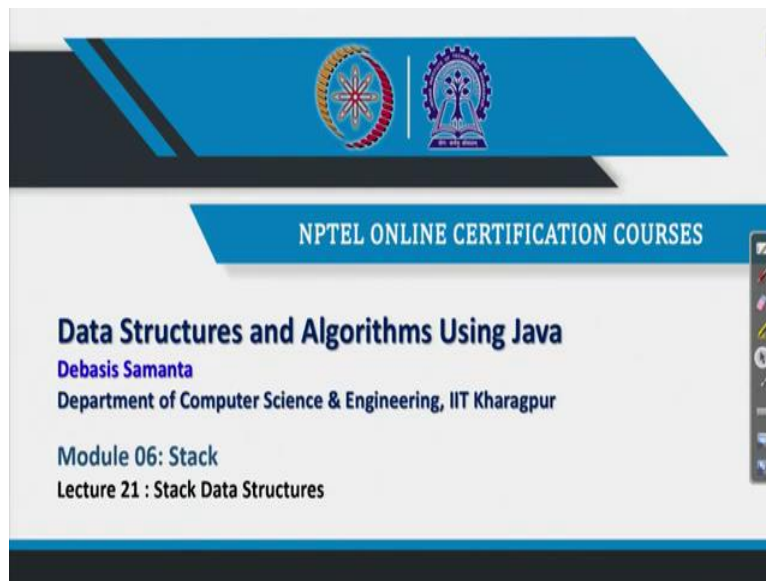


**Data Structures and Algorithms Using Java**  
**Professor Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 21 - Stack Data Structures**

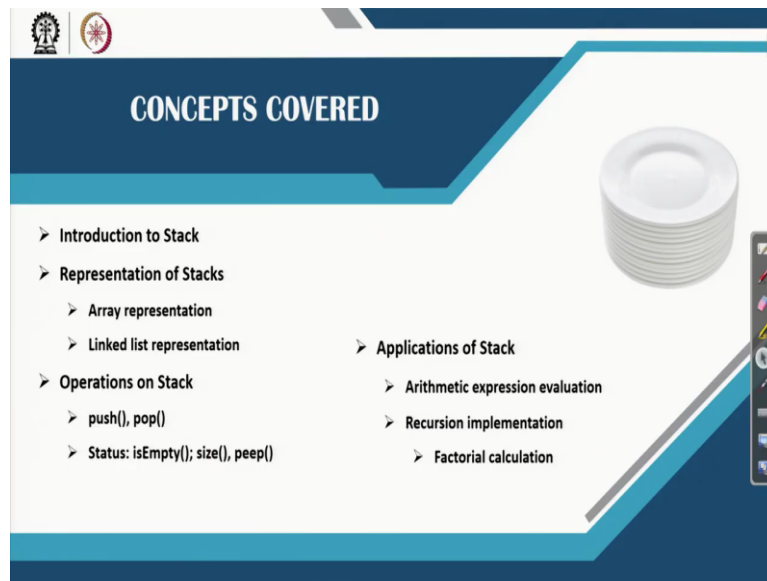
We shall plan our next module, the next module includes another important data structure, it is called the stack. Stack has many applications in programming. Without using stack in fact you cannot implement many algorithms actually.

(Refer Slide Time: 00:43)



So, today we have a basic idea about the stack structure and at the end of this lecture, I will try to give some idea about its few application. And then we will, in this module we also discussed in as a series we can discuss about how the programming for stack using Java programming can be done, and also how the stack can be implemented using Java collection framework. So this is the topic that we are going to cover.

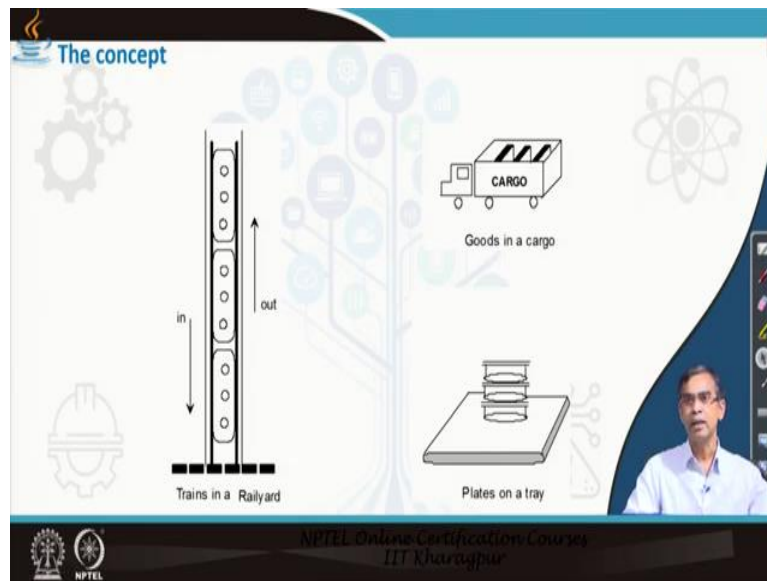
(Refer Slide Time: 01:20)



First we will introduce the stack concept and then, the how the stack can be stored in a memory. So there are two ways using array and using linked list that we will cover quickly. And obviously knowing a data structure means whatever the different operations that can be possible we will try the different operations on stack and finally try to understand about application of stack. So this is basically plan for this video lecture.

(Refer Slide Time: 01:44)





Now, let us come to the introduction. And then stack is basically very common in our daily life. For example, if you visit any restaurant then you can give an order for different plates and then the boy can serve you and usually boy can stack the different plates. So, boy first place the stack this one containing some item, then next and so on. Now, when the boy will serve to the customer so, basically the he will serve in opposite order the way he place the item. So, this will given first, then next one and next one.

So, this is basically the way the stack works for the restaurant people or boy. Now, here again loading and unloading a cargo basically follow the stack mechanism. So, basically you suppose load the car here in this cargo so, if you load in this way then while you are unloading, the unloading will be see the last which was loaded will be unloaded first. Now there is another example that is the example where the different rail, train, (bud) coaches are to be inserted into the rail yards.

So, this is the one train, this is another train and another train. So, if we, those trains are stacked into the rail yard and then while these trains are moving, you can see the train which added into the rail yard last is basically remove first. Now, all these concept actually if you see, the concept is that last in first out. So, that is why the stack data structure is also alternatively called LIFO structure because, last in first out. The element which are added at the end, will be removed at the first actually.

(Refer Slide Time: 03:35)

The slide is titled "Definition of stack". It contains a bullet point: "A stack is an ordered collection of homogeneous data element where the insertion and deletion operations take place at one end only." Below the text is a diagram of a stack. It is a vertical container with four items labeled "ITEM 1", "ITEM 2", "ITEM 3", and "ITEM 4" from top to bottom. An arrow labeled "TOP" points to "ITEM 1". Above the stack, an arrow labeled "PUSH" points down into the stack, and an arrow labeled "POP" points up away from the stack. The bottom of the stack is labeled "Bottom". To the right of the diagram, it says "Stack follows LIFO (Last In First Out) property". In the bottom right corner, there is a small video inset of a man speaking. The slide footer includes the NPTEL logo and the text "NPTEL Online Certification Courses IIT Kharagpur".

Now let us see what is the official definition of the stack. Now, stack like array, linked list as we have learned about is an order collection and also homogeneous because, a stack should includes only one type of data not the different type. So, if you want to maintain a stack of integers, then only you can add integer value but, not the other type of value. So, it is that is why it is called the ordered collection because, the order is maintained. Order means in which order you add, in that order it will be maintains.

So, it is not the random order and, and then, it basically allow one (op), all operations not from anywhere like array or linked list. In case of array you can insert anywhere, you can delete from anywhere. In the linked list also you can do the same thing. But in case of array your insertion should be at one place and deletion also from the same place so, one end only. As an example you can see, if this is the stack there and this is the top means is the current location where, the recent insertion takes place or deletion takes place.

Now, if you go for further insertion it is called the push in case of stack, or remove one called the pop operation, then it will takes place from there. In other words if you want to insert item 3, or insert, delete item 3, you will not be able to do that. Or if you want to insert some item, after item 4 you cannot do that. It will be always insert here. So, this is why insertion, deletion can be taken place at one place only. Now, if we do this it basically satisfy LIFO property, LIFO property is ensured by allowing or by restricting insertion, deletion of any other item or objects into the stack structure at one end only.

(Refer Slide Time: 05:34)

The image shows two slides from an NPTEL presentation. The top slide is titled "Representations of Stack" and features a coffee cup icon. The bottom slide is titled "Memory representations" and compares "Array representation" and "Linked list representation" with diagrams. Both slides include the NPTEL logo and a video feed of a presenter.

Okay so, this is the definition of a stack. I hope you have understood about the concept of stack, now let us see how a stack can be stored in a memory. There are two different way the stack can be maintained, one usual way this is little bit easy way you can say is the array representation or the exactly the array representation is, so basically, this is the array of elements and here you can see there are two locations, this is the bottom and this is the top.

That means array will grow from here only. So, we first we shall, we shall add element into the 0th index or 1 index or L index whatever it is there and then if you want to add next item in this way. So, if we go on adding element so, stack will grow in this direction. So, stack go in this direction. And at any instant the topmost element is pointed by one pointer, it is called the top. So, top indicates that which is the top most element in the stack.

And then stack can grow till it basically exhausted that means is, exhausted means if the element come at this location and top is here, and if you want to add further then, you will not be able to, because in that case it is called the stack is overflow. Now, so this is the array representation in case of array representation as you know size of the stack is limited. After sometimes if you go on adding so, it become overflow so it is just some sort of static way the array, the stack can be maintained.

Alternative to this array the linked list as you know, linked list supports dynamic, you can go on adding as many as you want so, there will be never stack overflow situation in case of array. Unless, the memory is exhausted. So, memory is exhausted means your memory bank is not able to supply to add a new node into your linked list. Now, so this way here again a very pretty simple as I told you that insertion or deletion may namely push and pop operation in case of stack it is called. They can be done only at one end, so this is basically at the front only.

So, top actually a pointer it points the first element in the, in the list. And if you want to add it, it is basically insert front and if you want to remove an element it is basically delete end, a delete front actually. So, only two operations that is basically applicable for the linked list representation of stack is basically insert front and delete front. And this way you can maintain a stack. So, this is the two broad ways of maintaining an array in a memory it is called the (linked) stack representation.

(Refer Slide Time: 08:26)

The image shows a presentation slide with a white background and a blue header and footer. The title "Operations on Stack" is written in blue text in the center. To the left of the title is a brown coffee cup icon with steam rising from it. The background features a faint tree-like structure with various icons (gears, a lightbulb, a book, a person) and a blue atom symbol in the top right. In the bottom right corner, there is a small video feed of a man in a light blue shirt. The footer contains the NPTEL logo and the text "NPTEL Online Certification Courses IIT Kharagpur".



**Operations on stack**

- Push** To insert an item into the stack
- Pop** To remove an item from a stack
- Status** To know the present state of a stack
  - if stack is empty,
  - size of the stack,
  - the element at top

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us learn about operation on stack. We have told that only two operations major operations those are possible is called inserting an item into a stack and removing an item into a stack. And there are few more operations are also possible to know, how many elements are there? What is the current size of the stack? Whether stack is empty or not? And which is the element at the top without any popping we can if want to show.

So, these are the method, these are, these basically methods are called miscellaneous methods which comes under the status operation. So, mainly three categories of operation called, push, push is basically inserting an item into the stack and removing an item basically called the pop and status is basically to check the size, empty (condi) state, or many other things that you can think about it.

(Refer Slide Time: 09:19)

**Stack with Array Representation**

NPTEL Online Certification Courses  
IIT Kharagpur

**Operation: Push with array**

We have assumed that the array index varies from 1 to SIZE and TOP points the location of the current top-most item in the stack. The following algorithm defines the insertion of an item into a stack represented using an array A.

**Input:** The new item ITEM to be pushed onto it.  
**Output:** A stack with a newly pushed ITEM at the TOP position.  
**Data structure:** An array A with TOP as the pointer.

**Steps:**

1. **If** TOP  $\geq$  SIZE **then**
2.     **Print** "Stack is full"
3. **Else**
4.     TOP = TOP + 1
5.     A[TOP] = ITEM
6. **Endif**
7. **Stop**

NPTEL Online Certification Courses  
IIT Kharagpur

Now, so let us see how the different operations can be implemented using each representation namely array representation and linked list representation. First we shall discuss about array representation and then we will come to the discussion of same thing what with linked list representation. Now the (staray) stack array representation is very simple, I have given an algorithm for you and I assume that the maximum size of the stack is decided by the field, it is called the size. So, stack will go from 0 to size minus 1, or 1 to size depending on whatever the way you want to index.

So, in case of Java it should be 0 to size minus 1. Now, here is the method how we can add into the stack. In this case I assume that index is from 1 to size. So, if top is greater than equal to size, so that means it reaches to the end, that means it is the stack overflow condition so, stack is full. Otherwise it will just move the top into the next position and add the item into that position and this completes the push operation in this case. So, this is basically how an item can be added into the stack. So this basically gives a screenshot about how the push before stack and then after stack.

As you can see the stack grow in this direction, right in this direction, starting from this is called the bottom of the stack, the first element of the stack and this is the current location, and if you add one element you see the top will be move to this and current location will be stored here. So, this is a pretty simple so, for the array you are basically adding some element, inserting some element into an array, and you can go on, you are able to do until array is not exhausted. So, this is the idea about how the push operation or insertion of an item into a stack using array can be done.



(Refer Slide Time: 11:20)

**Operation: Pop with array**

The following algorithm defines the deletion of an item from a stack represented using an array A.

**Input:** A stack with elements.  
**Output:** Removes an ITEM from the top of the stack if it is not empty.  
**Data structure:** An array A with TOP as the pointer.

**Steps:**

1. If  $TOP < 1$  then
2. **Print** "Stack is empty"
3. **Else**
4.  $ITEM = A[TOP]$
5.  $TOP = TOP - 1$
6. **EndIf**
7. **Stop**

Diagram illustrating the pop operation:

Before Pop(): The stack is represented by an array A with indices 1 to N. The TOP pointer is at index 1. The stack contains elements at indices 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. The TOP pointer is at index 1. A red arrow points to the element at index 1, labeled "Pop operation is failed".

After Pop(): The stack is represented by an array A with indices 1 to N. The TOP pointer is at index 2. The stack contains elements at indices 2, 3, 4, 5, 6, 7, 8, 9, 10. The TOP pointer is at index 2. A red arrow points to the element at index 2, labeled "Pop operation is failed".

NPTEL Online Certification Course  
IIT Kharagpur

Now, let us come to the another opposite operation is called pop. Pop is basically opposite to the push operation, push operation add element, pop operation on the other hand remove element. Now, you can continue this removal until stack is empty. So, before going to push operation it is your step is to check, whether stack is empty or not? Like before insertion you have to check whether stack is full or not. So, the starting statement that can be here if TOP less than 1. So, TOP less than 1 means it exceeds 1, I mean rather bottom location so, it goes there.

So, in this case if top is here this indicates that the stack is empty. Now, otherwise if top is less than this 1, that means some elements are there may be top is here. So, if top is here, it will basically delete this element and then top will be move to the next element. So, this will be the, this 1, for example here. So, this is the code here first if top is permissible, so that stack is not empty, then we can read the element and then top can be decremented by 1. So, that it can move to the next, it can point to the next item in the stack. so, this is a very pretty simple pop operation for a stack using array.

(Refer Slide Time: 12:40)

**Operation: Status with array**

The following algorithm tests the various states of a stack such as whether it is full or empty, how many items are right now in it, and read the current element at the top without removing it, etc.

**Steps:**

1. If  $TOP < 1$  then
2. **Print** "Stack is empty"
3. **Else**
4. If  $(TOP \geq SIZE)$  then
5. **Print** "Stack is full"
6. **Else**
7. **Print** "The element at TOP is",  $A[TOP]$
8.  $free = (SIZE - TOP)/SIZE * 100$
9. **Print** "Percentage of free stack is",  $free$
10. **Endif**
11. **Endif**
12. **Stop**

NPTEL Online Certification Course  
IIT Kharagpur

Now, there is other operation like status. So status operation is basically say whether stack is empty or if it is what is the total number of elements present in there, what is the capacity size, like, like, like. So, here basically if we see if top less than 1, stack is empty status you can print and if top greater than equal to size, you can say stack is full. So, empty and full, you can write (13:03) other methods also, I mean operation or in one operation you can do. In this example we have done all the things in one operation, called the operation is status.

Now, here also you can see how many total, I mean available things is available there so, we can see percentage calculation and so many things. These are pretty simple thing that you can do minimum calculation that what is about the stack size, and then capacity, availability, like this one. So, these are the three different methods, fundamental methods basically we need to have its applications in many context. Anyway application will be next discussion, we will consider first.

(Refer Slide Time: 13:39)

**Stack with Linked List**

**Operation: Push with linked list**

*Input:* ITEM is the item to be inserted.  
*Output:* A single linked list with a newly inserted node with data content ITEM.  
*Data structure:* A single linked list structure whose pointer to the header is known from STACK\_H and TOP is the pointer to the first node.

**Steps:**

1. `new = GetNode(NODE)`  
*/\* Insert at front \*/*
2. `new->DATA = ITEM`
3. `new->LINK = TOP`
4. `TOP = new`
5. `STACK_H->LINK = TOP`
6. **Stop**

**Before push()**

**After push()**

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us come to the dissection about stack with linked list representation, which is more I mean advantageous than the array representation, because array representation has suffers from memory, I mean memory constant actually. That means you cannot go beyond certain size or whatever it is there. But linked list can go beyond size, I mean it did, it does not depend on the size, until the memory is exhausted. Memory means the actual memory where your program data everything is stored.

Now, let us see how the array representation of the things. I told you that in case of linked list representation the insertion and deletion that mean push and pop operation will takes place from the front only. So, here the push operation, the scenario how it will look like this. So,

this is suppose the linked list at any instant it can be empty of course, if it is not empty let us see how the push operation can be done. As I told you push operation should be.

So, this top actually indicates that which is the top most element in the linked list now if you want to add a new element what exactly that, okay this link is not correct. actually this is the mistake in this diagram. You should ignore this one, it basically, so it basically, okay new node, this is the new node obtained it and this is the top. Okay, so this is the top most element. So, the new node that can be there you can add into here. That is all.

And then this pointer will be mention there. So, this way you can go on. So it is just okay, we have already learned about how element a new node can be added at the front of a linked list. So, that concept is there and here is the programming code you can follow it. Now how an element can be added into the linked list. We are already learned about the linked list is insertion and everything, so we just follow the same thing here and again. No need to discuss in details about it actually.

(Refer Slide Time: 15:39)

**Operation: Pop with linked list**

**Input:** A stack with elements.  
**Output:** The removed item is stored in ITEM.  
**Data structure:** A single linked list structure whose pointer to the header is known from STACK\_H and TOP is the pointer to the first node.

**Steps:**

1. **If** TOP = NULL
2. **Print** "Stack is empty"
3. **Exit**
4. **Else**
5. ptr = TOP→LINK
6. ITEM = TOP→DATA
7. STACK\_H→LINK = ptr
8. TOP = ptr
9. **EndIf**
10. **Stop**

The diagram illustrates the pop operation in three states: **Before Pop()**, **After Pop()**, and **Failed Pop()**. In the 'Before Pop()' state, a linked list has a 'Header' and a 'TOP' pointer pointing to the first node. In the 'After Pop()' state, the 'TOP' pointer is updated to point to the second node, and the first node is circled in red. In the 'Failed Pop()' state, the 'TOP' pointer is NULL, indicating an underflow.

So, like push, the pop operation is like this also and so this is basically if the linked list given to you before pop top is basically the top most element then, if we want to do the pop, the after pop actually is basically return this element, it is basically remove. Remove the first element and then pointer will be point to this one. So, this is the user practice and if the list is empty, obviously if you perform pop, it basically shows underflow or top stack is empty like this one.

And the simple code it is mentioned here, for your understanding. This is so simple that you do not required any further discussion, if you read it, you will be able to understand it. So, I do not want to okay spend time on this. Now, so we have learned about push and pop operations on the linked list. Similarly, the status also whether it is empty or it is, there is no concept of full actually.

(Refer Slide Time: 16:33)

**Operation: Status with linked list**

**Input:** A stack with elements.  
**Output:** Status information such as its state (empty or full), number of items, item at the TOP.  
**Data structure:** A single linked list structure whose pointer to the header is known from STACK\_H and TOP is the pointer to the first node.

**Steps:**

1. ptr = STACK\_H → LINK
2. If (ptr = NULL) then
3.   Print "Stack is empty"
4. Else
5.   nodeCount = 0
6.   While (ptr ≠ NULL) do
7.     nodeCount = nodeCount + 1
8.     ptr = ptr → LINK
9.   EndWhile
10. Print "Top item:", TOP → DATA,  
    "Size = ", nodeCount
11. EndIf
12. Stop

NPTEL Online Certification Course  
IIT Kharagpur

So, we can understand whether it is empty or not or what is the total number of elements present there or not? All those things you can have it from the stack actually. So, there is a code actually given, it is a pseudo code, it is the algorithmic level code it is there. Just for an understanding about I have given some idea about that how the code will look like, and if this is the code, just similar to the array also, but in this case it is using linked list only.

So, if you are familiar to linked list, its operation, its performance, writing program for you, it will be very easy actually. And I should advice you to go on writing the program for a stack also using linked list. But, you will see exactly how better programming can be done in another video. Okay so, this is about a stack in representation, we have learned about two type of representation, array representation and linked list representation, and how the different operation for each representation can be done.

(Refer Slide Time: 17:33)



Now, we will discuss about applications of stack. Now these applications, actually that is why stack is so popular. Now, stack is so popular because (there), there are many algorithms if you want to build, develop, for your application where the stack needs to be there. Now, there are many sorting algorithm, for example of searching algorithm or some calculation like factorial, in many (situ) occurrence, many situations you have to use stack.

But one thing that stack is so, I mean frequently occurred one data structure, that is why many operating system, or compiler they have their own OA of managing stack automatically, without any headache of the programmer. This for example, if you want to use a stack for your some calculation say factorial calculation in a recursive, for any recursive programming if you know stack is the must. So, without any stack you can, but when you have, possibly you have, you are familiar to many stack and many methods which you declare using recursion.

But, in that case if you see you did not use stack explicitly rather. Implicitly, you call method recursively but, whenever the recursive method is executed, then there is a run time manager automatically maintain a stack and do the all operation like push, pop of their own. So, at the back end actually stack is working actually but, we do not have to bother about it. This is the good things that support is given to the programmer actually. Anyway, but see, if you want to use it for some operation, where you want to explicitly use stack how it can be done.



(Refer Slide Time: 19:24)

The slide, titled "Some common applications", lists several uses of a stack in system programming. It includes a bulleted point at the top and four numbered sections, each with sub-points. A small video inset of a speaker is visible in the bottom right corner of the slide area.

- Stack is extensively used in **system programming**
- 1. Evaluation of arithmetic expression
  - Conversion an expression to postfix notation
  - Evaluation for the value
  - Machine code generation
- 2. Execution of recursive call of a function
  - Factorial calculation
  - Tower of Hanoi
  - Quick sort
  - Merge sort
- 3. Scope rule implementation
  - Static and dynamic scope rule management
  - Activation record management
- 4. Memory management
  - Run-time memory management
  - Code generation
  - Syntax parsing

NPTEL Online Certification Course  
IIT Kharagpur

And before going to this thing, I list few major applications where the stack is extensively used. One used is that using arithmetic expression. In your program you can write many arithmetic expressions for your mathematical calculation or whatever it is there. And ever have you thought, how compiler or run time manager calculate your expression? Now, this is not an easy task that okay an expression will be calculated so easily. But, to do these things run time manager consider an stack to follow it.

Now, this is the one example, next recursive call, implementation, execution of a recursive function, they are also the stack needs to be applied. For example, a factorial calculation, quick sort, merge sort, then tower of Hanoi problem, there are many, GCD calculation, what is not. Many methods we usually like to have it using recursion. So, that also you can see how the stack can be used to support your recursive call execution then scope rule implementation, this is one compiler issue.

So, if you want to pass your programming language, and then in Java for example, there is a dynamic binding, static binding, static scope, dynamic scope, run time polymorphism, so many things are basically implemented. Program, you go on writing the program but, run time manager check that which scope rule should be binding to a particular object or variables or field. And then accordingly it will resolve it. So, this scope resolution is basically done by means of that is called stack, in that case that stack is called activation record management.

Now, there are many applications like and finally the last but not least is the memory management. Now, memory management. In many situation it is again related to the compiler execution or a system level programming, processing or natural language processing or machine learning information processing. So, there you have to follow the stack actually. Now, I have listed only few. Actually I could go on incusing more application in this category.

Okay anyway, these are the important applications that you can, you can I mean that we are usually doing it whenever a programmer has to develop application. Maybe system application, system level application or application for a business organization. Whatever it is there. Anyway so, in this lecture it is not possible to discuss all application because, time is short. But, we can consider at least a very simple application. I will just discuss about evolution of arithmetic expression quickly.

But, here we will see exactly, in order to evaluate an arithmetic expression. so there are two task actually. The first is that we have to convert the given expression into a specific notation, this is called the post fix notation. And then expression which we are familiar with, it is called the infix notation. So, basically the first step is that infix to postfix then only we can go for calculating, I mean evaluating an expression and finally evaluation for a given value of a unknown or variables.

(Refer Slide Time: 22:41)



Application: An arithmetic expression

$$A + B * C / D - E \wedge F * G$$

Precedence and associativity of operators

Operators	Precedence	Associativity
- (unary), +(unary), NOT	6	-
^ (exponentiation)	6	Right to left
* (multiplication), / (division)	5	Left to right
+ (addition), - (subtraction)	4	Left to right
<, <=, +, <>, >=	3	Left to right
AND	2	Left to right
OR, XOR	1	Left to right

NPTEL Online Certification Courses  
IIT Kharagpur

Now, quickly we will discuss about the arithmetic expression evaluation. Now here is an example, first let us see this is an typically an arithmetic expression look like. Now, why evaluation is not so easy? This is not easy because, this expression is given to say 5 different students for the same values of ABCDEFG you may find many results from the many student, and every student can claim that yes my calculation is correct.

Actually, which result is correct? So, that is basically idea is that whenever you have to evaluate an expression, you have to consider two things about the operators there in the expression. Operator means here for example, addition is an operator, multiplication, subtraction, division, exponential these are the operation. Now, so these are the operators. Now what is the problem, so far an arithmetic expression evaluation is concerned is that, all operators are not of equal precedence.

What is the idea? If suppose A plus B star C it is given then you should perform multiplication first then come to the addition. So, this is called the precedence. So, if many operators are there, operations are to be done then you have to do, that operator which has the highest precedence. So, here I have listed the precedence of operation, as you can see the different operator logical operator, logical operator, bullion operator, arithmetic operator, and XOR, they are basically low precedence actually.

And here unary operator is the highest precedence, exponential operator is there. So this basically list the different order of precedence of the different operator those are there usually available in arithmetic expression. Other than precedence there is one is called associativity. Now, some calculation you know you have to do, in left to right, and something is right to

left. Yes, so now usually if, if you see the many operator that we have mentioned they are usually left to right, calculation. But, there are some operators is the right to left.

For example, if I write so, E then exponential F then exponential again X. So, E to the power F to the power X, okay? So it is like this. So, E to the power F, then F to the power X. so, basically we have to first do it and then we have to do it. So, F to the power X should be calculated first, then we can come to the E to the power X. So, it is like 2 to the power M then power N. So M to the power N first, then 2 to the power, M and M to the power N it is like this.

So, this is basically called the right to left now here exponential operation you see right to left. On the other hand actually this associativity usually coming into the picture to resolve if the operators are having the same precedence but, different operators. For example, if I write say A plus B minus C. Now, here you see, the operators here plus and minus, they are of same precedence. But, for plus and minus if you see, they are basically left to right, this means that we have to calculate first A plus B then you have to calculate the next one. So, basically left to right order, we have to the calculation.

Now, so if you follow all these precedence rule and associativity rule, in this expression then only your calculation is there. In any order if you do the operation then it may gives the wrong result for the same values. So, that is why 5 student can come with the different results because, they do not follow this (assoc) associativity and precedence properly. Now, so this is the problem that is why arithmetic evaluation expression is not an easy task or trivial task actually.

(Refer Slide Time: 26:57)

**Application: An arithmetic expression in parenthesized form**

$A + B * C / D - E ^ F * G$

Operator	Precedence	Associativity
~ (unary), ~ (unary), NOT (exponentiation)	6	Right to left
* (multiplication), / (division)	5	Left to right
+ (addition), - (subtraction)	4	Left to right
<, <=, >, >=	3	Left to right
AND	2	Left to right
OR, XOR	1	Left to right

$A + B * C / D - E ^ F * G$

1 2 3 4 5 6

NPTEL Online Certification Courses  
IIT Kharagpur

Now, this see how this can be done here. I have given some example here, in which order the different expression is there. Because it is the highest precedence, so, you have to first calculate this one. So, this is the first one and then this is the second one, and this is the different order that we can do it as it is shown there. So, this basically the order according to this associativity and precedence rule for the same order it can be calculated. And if we follow this rule then it will give you the correct result. So, this way we have to have the mechanism, so that how the precedence and associatively should be followed while you are calculating or evaluating an arithmetic expression.

(Refer Slide Time: 27:39)

**Application: An arithmetic expression in FORCED parenthesized form**

$((A + B) * ((C / D) - (E ^ (F * G))))$

$((A + B) * ((C / D) - (E ^ (F * G))))$

1 2 3 4 5 6

NPTEL Online Certification Courses  
IIT Kharagpur

Now, there are another way also, some expression can be parenthesized. If you the parenthesize then you know, all the innermost parenthesis should be calculated first. So, this kind of things if you want to overrule about the default precedence and associativity over the given an expression it is like. So, this is an example here what you can see. We have overruled the precedence by inserting this one. So, in that case as you can see these are the operation should be taken first because, these are the innermost.

Then once it is done then this operation will taken first and then this operation and then final this operation. So, in which order all these things are there it is shown here. You can check it. So, if we have the FORCED parenthesis or expression is parenthesize then parenthesize can also, can tell you that in which order you should do the expression, evaluate the expression. So, that is why we have to have the mechanism.

(Refer Slide Time: 28:36)

In-stack and in-coming priorities of symbols

Symbol	In-stack priority value	In-coming priority value
+ -	2	1
* /	4	3
^	5	6
operand	8	7
(	0	9
)	-	0

And this mechanism can be followed if we follow, stack there. Now, in order to have the stack into there is basically check, whether an operator should be followed according to this precedence and associativity. So, they are basically for different (order) order, here the programmer mathematization has defined two priorities called ISP, in stack priority and ICP called in coming priority. So, for different symbols they have the different priority. It is decided empirically.

So, you have to keep in your memory while you are calculating. So, here idea is basically is that, idea is basically that, whenever we can pop and push, that decide by the in stack priority and in coming priority. If some item which we are going to have in our calculation and whether we perform some operation or not depending on in stack priority and in coming



priority. So, there is a (possib) rule and based on which rule an algorithm is basically designed.

(Refer Slide Time: 29:33)

**Application: Conversion of an infix expression to postfix expression**

*Procedure to perform expression conversion*

Function	Description
ReadSymbol ( )	From a given infix expression, this will read the next symbol.
ISP (X)	Returns the in-stack priority value for a symbol X.
ICP (X)	This function returns the in-coming priority value for a symbol X.
Output (X)	Append the symbol X into the resultant expression.

NPTEL Online Certification Courses  
IIT Kharagpur

**Application: Conversion of an infix expression to postfix expression**

**Input:** E, simple arithmetic expression in infix notation delimited at the end by the right parenthesis ')', incoming and in-stack priority values for all possible symbols in an arithmetic expression.  
**Output:** An arithmetic expression in postfix notation.  
**Data structure:** Array representation of a stack with TOP as the pointer to the top-most element.

**Steps:**

1. **TOP = 0, PUSH('(')** // Initialize the stack
2. **While (TOP > 0) do**
3.     **item = E.ReadSymbol( )** // Scan the next symbol in infix expression
4.     **x = POP( )** // Get the next item from the stack
5.     **Case: item = operand** // If the symbol is an operand
6.         **PUSH(x)** // The stack will remain same
7.         **Output(item)** // Add the symbol into the output expression
8.     **Case: item = ')',** // Scan reaches to its end
9.         **While x ≠ '(' do** // Till the left match is not found
10.             **Output(x)**
11.             **x = POP( )**
12.     **EndWhile**

NPTEL Online Certification Courses  
IIT Kharagpur

Application: Postfix notation of an arithmetic expression

$$((A + ((B \wedge C) - D)) * (E - (A/C)))$$

$$(A + ((B \wedge C) - D)) * (E - (A/C))$$

$A+B$   
 $AB+$

$$ABC^{\wedge}D-+EAC/-*$$

And then algorithm is mentioned here. This basically is an algorithm shows you how you can convert an infix expression to postfix expression. Now before going to this infix and postfix expression, I just want to have a quick discussion about what is the postfix expression actually. Now, the postfix expression is basically as you see, see suppose A plus B, if it is the addition of two operand A plus B and operator is basically in between the operand. So, that is why it is called the infix. The postfix operation is basically operator should be at the end of the operand, so AB plus. So, this is the concept of the postfix.

Now, here this is the expression in infix notation and how we can convert into the postfix notation, and this is the postfix notation. As you see, all the operator those are there with the operand they should go to the end. So, it is the BC, BC and then here BC minus D this way it is like this one. Now so, so this is basically one mechanism to be followed. Actually the idea is that if the postfix operation is known to you, then calculating the value of an expression and using stack is far easier, so that is why we have to do it.

(Refer Slide Time: 31:05)

**Application: Conversion of an infix expression to postfix expression**

**Input:** E, simple arithmetic expression in infix notation delimited at the end by the right parenthesis ')', incoming and in-stack priority values for all possible symbols in an arithmetic expression.  
**Output:** An arithmetic expression in postfix notation.  
**Data structure:** Array representation of a stack with TOP as the pointer to the top-most element.

**Steps:**

1.  $TOP = 0$ ,  $PUSH('')$  // Initialize the stack
2. **While** ( $TOP > 0$ ) **do**
3.  $item = E.ReadSymbol()$  // Scan the next symbol in infix expression
4.  $x = POP()$  // Get the next item from the stack
5. **Case:**  $item = \text{operand}$  // If the symbol is an operand
6.  $PUSH(x)$  // The stack will remain same
7.  $Output(item)$  // Add the symbol into the output expression
8. **Case:**  $item = '('$ , // Scan reaches to its end
9. **While**  $x \neq '('$  **do** // Till the left match is not found
10.  $Output(x)$
11.  $x = POP()$
12. **EndWhile**

NPTEL Online Certification Courses  
IIT Kharagpur

**Application: Conversion of an infix expression to postfix expression**

13. **Case:**  $ISP(x) \geq ICP(item)$
14. **While** ( $ISP(x) \geq ICP(item)$ ) **do**
15.  $Output(x)$
16.  $x = POP()$
17. **EndWhile**
18.  $PUSH(x)$
19.  $PUSH(item)$
20. **Case:**  $ISP(x) < ICP(item)$
21.  $PUSH(x)$
22.  $PUSH(item)$
23. **Otherwise:**  
 $Print$  "Invalid expression"
24. **EndWhile**
25. **Stop**

NPTEL Online Certification Courses  
IIT Kharagpur

Okay, now let us come to the algorithm that you can think about how this is an algorithm. I will not be able to discuss details, you can take your own time to understand how this is pretty simple, when to do the push and when to pop. So, it depends on it like this. So, read a, read a or read a symbol from the expression. The symbol may be operator or operand or brackets.

And then, it basically either push or pop depending on that condition. So, if you follow this algorithm and write your code you will be able to easily (evaluate) convert an infix expression to postfix expression. I just, okay while I will do the programming that time I will discuss about it. But I kept it pending for you to consult it in either any book or you can

follow this slides also, you can follow it. So, this algorithm is, okay as the time is little bit required, so to understand this one.

(Refer Slide Time: 31:59)

**Application: Conversion of an infix expression to postfix expression**

13. **Case:**  $ISP(x) \geq ICP(\text{item})$
14. **While**  $(ISP(x) \geq ICP(\text{item}))$  **do**
15. **Output**(x)

**Note:** This is a procedure irrespective of the type of memory representation of the stack to convert an infix expression to its postfix form using the basic operations of the stack, namely PUSH and POP. These operations can be replaced with their respective versions and hence implementable to stack with any type of memory representation.

23. **Otherwise:**  
**Print** "Invalid expression"
24. **EndWhile**
25. **Stop**

NPTEL Online Certification Courses  
IIT Kharagpur

So this is about the infix to postfix expression calculation. Now as I can say that, okay so stack is used to here to follow the ordering of the different operator or ensuring the precedence and associativity of the different operators in the expression calculation and by virtue of push and pop operation.

(Refer Slide Time: 32:21)

**Application: Evaluation of a postfix expression**


*Input:* E, an expression in postfix notation, with values of the operands appearing in the expression.  
*Output:* Value of the expression.  
*Data structure:* Array representation of a stack with TOP as the pointer to the top-most element.

NPTEL Online Certification Courses  
IIT Kharagpur

### Application: Evaluation of a postfix expression

**Steps:**

1. Append a special delimiter '#' at the end of the expression
2. `item = E.ReadSymbol ( )` // Read the first symbol from E
3. **While** (item  $\neq$  '#') **do**
4.   **If** (item = operand) **then** // Operand is the first push into the stack
5.     **PUSH**(item)
6.   **Else**
7.     `op = item` // The item is an operator
8.     `y = POP ( )` // The right-most operand of the current operator
9.     `x = POP ( )` // The left-most operand of the current operator
10.     `t = x op y` // Perform the operation with operator 'op' and operands x, y
11.     **PUSH**(t) // Push the result into stack
12.   **EndIf**
13.   `item = E.ReadSymbol ( )` // Read the next item from E
14. **EndWhile**
15. `value = POP ( )` // Get the value of the expression
16. **Return**(value)
17. **Stop**



NPTEL Online Certification Courses  
IIT Kharagpur

So, this is basically the main idea about it and once the postfix expression is known to you, then you can write another algorithm where input will be postfix expression and then output will be the value of it depending given the different values of the operator. So, this is an algorithm that again you can follow how the postfix expression can be evaluated for given set of, values of the operand. So, this algorithm again I will not be able to discuss in detail because it will take much.

But, I advise you to go through the code, I mean every line you can follow. That mean in which time we have to the push, pop and push, it is basically the algorithm is written that way. So, this algorithm is very well exercised algorithm actually and you can find this algorithm in many places and you can follow and you can write the code accordingly, following this algorithm. Once algorithm is known, then writing code is not the big job. So, you can follow and you can see that how it works there.



(Refer Slide Time: 33:14)

# Recursion Implementation

The slide features a central graphic of a tree with various icons (gears, a lightbulb, a book, a person) as branches. To the left is an icon of a coffee cup with steam rising from it. The background is a light blue and white color scheme.

### Application: Recursive function call

*Input:* An integer number  $N$ .  
*Output:* The factorial value of  $N$ , that is  $N!$ .  
*Remark:* Code using the recursive definition of factorial.

$$N! = 1 \times 2 \times 3 \times \dots \times (N-2) \times (N-1) \times N$$
$$= N \times (N-1) \times (N-2) \times \dots \times 3 \times 2 \times 1$$
$$0! = 1$$

**Steps:**

```
1. If (N = 0) then // Termination condition of repetition
2. fact = 1
3. Else
4. fact = N * Factorial_R(N-1)
5. EndIf
6. Return(fact) // Return the result
7. Stop
```

### Application: Recursive function call

**5!**

**Steps:**

```
1. 5! = 5*4!
2. 4! = 4*3!
3. 3! = 3*2!
4. 2! = 2*1!
5. 1! = 1*0!
6. 0! = 1
7. 1! = 1
8. 2! = 2
9. 3! = 6
10. 4! = 24
11. 5! = 120
```

The slide shows the recursive steps for calculating 5!. Red arrows indicate the flow of the recursive calls and returns. The code block on the right is identical to the one in the previous slide.



Now, finally the recursion implementation and the recursion implementation as I told you, the stack is required. Why the stack is required, because this is a recursive method, recursive function you can see, how to calculate the factorial in sort of thing, this program. But, here the, here the implementation of this recursion is required because, as you see in order to calculate 5 factorial, in order to calculate 5 factorial, you have to compute 4 factorial.

So, 5 factorial cannot be finish until 4 factorial. In order to 4 factorial calculation so, so half way computed calculation need to be stacked and then go for 4 factorial. Now, again 4 factorial means 3 factorial, and this way you have to go. Now, when you come to the 0 factorial means it is the termination then, then we can just pop. So, this way actually go on pushing all the intermediate calculation, and then finally that popping, and finally you can get to the this one.

So, push and pop are basically call of the different methods whenever you call a method just go for pushing and whenever the next is running, I mean popping so, it is towards the final calculation. So, this basically idea is that you do not have, if you write a program following this algorithm, it will automatically use a stack your run time manager will do it. You do not have to use the stack and then push, pop and everything, but it is not your headache. So, for many calculation there the system level the stack already maintain by the operating system or run time manager actually.

(Refer Slide Time: 34:53)



Okay so, these are the few operations that we have discussed in this lecture. As it is not possible to discuss many other application, I would suggest you to follow this book. This book has many application, well exercised with illustration and all algorithms are there, all

algorithms obviously programming language independent way, (Java) it not necessary written in Java.

But you will be able to write the code, following Java syntax and everything. So, this is the stack and then okay fine we will discuss about stack programming in the next video lecture. Where we can see how the stack can be implemented using Java programming language, whatever the operations that we have mentioned using array or linked list can be, can be understood. Okay thank you very much.