Compiler Design

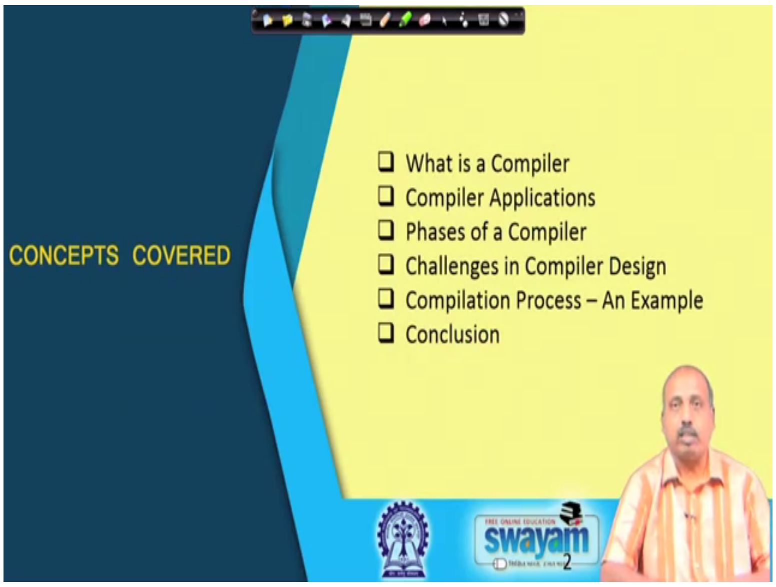
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 01
Introduction

So, welcome to this course on Compiler Design. So, in this course we will try to see like the different phases in the design of a compiler. But to start with it apparently seems very much difficult that given a language, how do it translate it into another language. But at the end of the course we will see that certain parts of it can be automated and certain parts of it depends on the experience and the expertise of the programmer, to develop the corresponding portions of the tools.

So, there are different parts of this course, while some parts of it have got formal background on automata theory, other part they are guided by few rules. For some of the programming language constructs the rules are well defined. Now, if we come across a new programming language, can we need to you need to redesign those rules ok. So, in this course we will try to have an overview of this entire process.

(Refer Slide Time: 01:24)

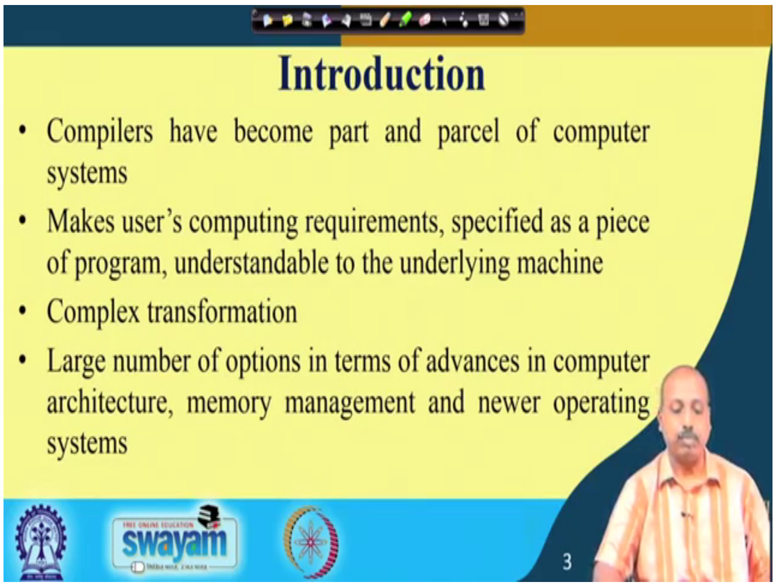


CONCEPTS COVERED

- What is a Compiler
- Compiler Applications
- Phases of a Compiler
- Challenges in Compiler Design
- Compilation Process – An Example
- Conclusion

So, to start with in this introductory lectures so, we will try to see like what do you mean by a compiler then, compiler applications, phases of a compiler, then challenges that we have in compiler design and compilation process. So, we will see an example by which we will try to illustrate like, what are the different phases of a compiler and we conclude the lecture by summarizing what you have what we have in the introductory part.

(Refer Slide Time: 01:50)



Introduction

- Compilers have become part and parcel of computer systems
- Makes user's computing requirements, specified as a piece of program, understandable to the underlying machine
- Complex transformation
- Large number of options in terms of advances in computer architecture, memory management and newer operating systems

So, to start with compilers have become part and parcel of today's computer systems. Now if we look back the early computer system that we are developed so, they have a huge computers in terms of the hardware and it required a good amount of space to house that particular computer.

Apart from that, say apart from these electrical and electronic challenges the other part of it was to make it work. Make it work means to make it do some useful competitions. So, for that purpose we needed to write a few write a programs and those programs are to be entered in some machine level language. So, as we know that computers ultimately they understand binary 0's and 1's. So, we need to write the program which is meaningful to the computer in terms of 0's and 1's. So, you can imagine like the amount of difficulty that you will have. So, if you have to write a even a 10 line program into (Refer Time: 02:52) convert it into binary 0's and 1's, which are meaning full to the computer.

So, that so early generation computers there this was the main problem and this translation process from this program that the program that is understandable by human being to a program understandable by the underlying computer was a big challenge, and it was done by means of manual methods by which we know the codes of each and individual instructions and we do translation by hand.

So, that made it very difficult and slowly the system improved and today we do not even understand that whatever we are writing and in high level language. So, it is not directly understandable by the by the computer, that is executing it. So, there are a lot of translations that are involved in the process. So, this translations are the essential part that we have in compilers. So, this compilers they make users computation requirements, which is specified by means piece of program understandable to the underlying machine.

So, user writes a program; that program maybe syntactically correct it is grammatically correct as per the language, and the program is hopefully syntactically correct, that is the meaning of the program is what the user wants to compute. For example, if you are trying to write a program which computes a roots of a quadratic equations, then once I write a piece of program in some language maybe in C or may be in some Pascal or C + +, Java whatever it is.

So, there are two types of mistakes that I could have done; one is I have some grammatical mistake that is made for example, in C language you know that every statement it ends with semi column. So, you may miss some semi column at some place so, that give some syntactically error. Maybe I am using a variable which is undefined which is not defined so, far. So, that is that me also be a that may be also an error. But there is a difference between these two types of errors. In the first type when I say that I missed a semi column. So, that is a grammatical error where when I say that I have not declared a variable. So, the machine does not know what to interpret for this particular variable, the type of whether it is integer real characters so, that is not understandable by the computer.

So, the type of operation that we are trying to do on the variable is not well defined. For example, if I have if I am having two integer variable, I can do addition operation, but if the variables are of type character array. So, I cannot do the edition I can only do string concatenation or say string operations on them. So, there is a basic difference between the two types of errors that we understood that which one is syntactical error another when the meaning of the program is not correct. So, that is the semantic error. So, it is a compiler the designer job to catch both of these types of errors syntactic error and semantic error. So, when the user has specified his requirement. So, then the requirement itself we need to analyze whether the requirement has been specified properly syntactically, and or and the meaning of the program is also clear. When everything is alright, then only the compiler will do a translation. So, it will translate the program from whether high level language to the machine level language and that includes complex transformation.

So, in this course we will see how those transformation are taking place. So this transformation will happen accordingly and with the increase in the complexity of the computer architecture and operating systems. So, the challenge that the compiler designers face so, that is also increasing. For example, like say today if you look into any advanced computer architecture system. So, it has got many interesting feature; one typical example is the CISC versus RISC architectures. So, CISC architectures we have got instructions which are pretty complex and the RISC architecture we have got instructions machine level instruction which are very simple. But the main difference is that, in case of RISC the instructions are of equal size and they take more or less same amount of time for execution whereas, for CISC architecture so, it is a other way. So, it takes the instructions are of variable size and they take different amount of time for their execution.

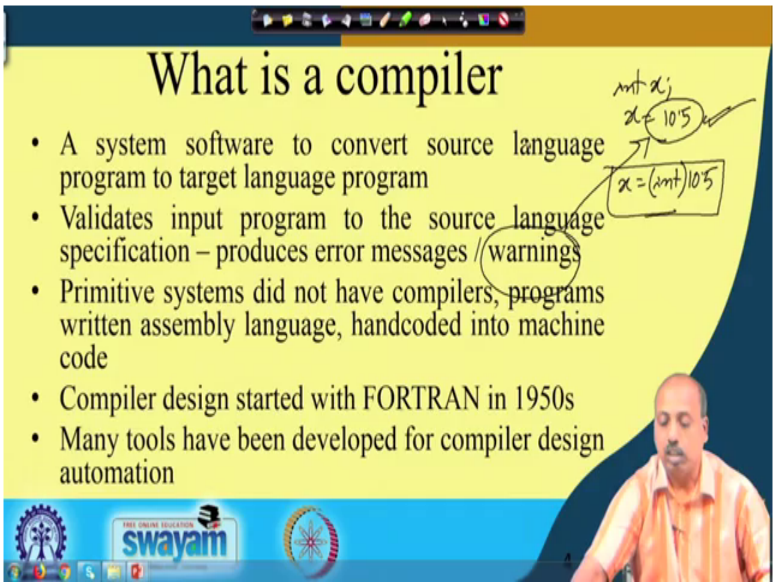
In fact, while you are trying to do some work so, if I give you instructions which are very simple in nature. So, you may possibly do that operation in a many more compact fashion. Like if I am trying to find the roots of a quadratic equation then, if I have very simple instructions in my hand, I can possibly do that operation without creating much of extra executions. On the other hand if I am having complex instructions so, some of the instructions may not be a very much necessary whole part of the instruction may not be complete necessary for some operations, but still I have to take it.

So, that way the CISC instruction they are going to be difficult to handle as far as the compiler designers are concerned and in fact, this movement from CISC to RISC this happened, because of this compiler designers from the compiler designers perspectives. So, it was suggested that if I have got simpler machine level instructions, it is easier to generate efficient code. So, with that this is done. So, with the advances in computer architecture so these challenges that are faced by the compiler designer so, that is going up.

So, memory management policies like today you know that almost all computers operating systems they are supporting virtual memory. So, virtual memory supported then so, we have to we have to judge like what are the most relevant operations that we are going to do, and possibly we want to keep those relevant portions in some on part of memory, which is not going to be swapped out to the secondary storage. So, that the operational efficiency is high.

So, this way this memory management plays an important role in determining the compilers generated codes performance. On the other hand the operating systems the new operating systems are coming. So, they are also having and new and newer features. So, accordingly or the challenges that are faced by the compiler designers so, they are also increasing significantly.

(Refer Slide Time: 09:39)



What is a compiler

- A system software to convert source language program to target language program
- Validates input program to the source language specification – produces error messages / warnings
- Primitive systems did not have compilers, programs written assembly language, handcoded into machine code
- Compiler design started with FORTRAN in 1950s
- Many tools have been developed for compiler design automation

int x;
$x \leftarrow 10.5$
$\boxed{x = (\text{int})10.5}$

So, what is a compiler? So, let us try to understand what is what is it. So, compiler is a system software, that converts source level language source level source level language program into target language program. The source language may be some high level language like say C, C ++, Java, FORTRAN, Pascal.

Now, how many different languages have been developed so, far that is innumerable. So, we cannot just go on giving examples, but we can say that. So, compiler for every language that he designed. So, if he want that the corresponding program will be executed by the underline computer. So, there must it must be translated into some machine level code, and that has to be done by the compilers. So, compiler is a system software, that converts source language program to target language program.

So, here the target language means in this particular case we are assuming machine language, but very shortly we will see that it is necessarily machine level language. So, it may be something else also. Second important thing that the compiler does is that it validates input program to the source language specification. So, source language any language is specified by means of its grammar for example, if we look into the English language. So, English language has got its own grammar ok. So, any English sentence that we write. So, you can check whether it is grammatically correct or not by properly analyzing the sentence.

So, similarly when you write a program in for some in some language, so, we can consult the grammar rules of the language and accordingly we can say whether the program is grammatically correct or not. So, if the program is not grammatically correct, then the compiler should produce some error messages. So, that such as for example, semi column is missing then it can say that one semi column is missing at this point.

Sometime we also need to produce warnings. So, warnings are like this that, at some places may be the programmer has not retained some specific declaration or specific transformation that is needed, but it may not be an error. So, if the program will execute but the out outcome of the program may be unpredictable. So, we will take a small example and we will take a small example like for example, in C language program suppose I write say integer x, and then x equal to 10.5. So, if we do this. So, there is a problem because this 10.5. So, this is not an integer number. So, this is a real number and x is an integer as I have defined.

So, you see that when the system will try to do this 10.5 assignment to x. So, what will happen it is very much system dependent, because the 10.5 being a real variable maybe it is given 6 bytes of space or eight bytes of space whereas, integer x being an integer variable, depending upon the system it may be given 4 to 6 bytes. So, what I essentially means is that, the space allocated to an integer variable and the real variable so, they are not same.

So, naturally when you do this type of assignment, the value that will be copied into x is not very clear it is not very much sure ok. So, if you and it can vary from one computer system to another computer system. So, output produced in one computer may be different from what is produced on the other computer. So, ideally I should write if I really want that this integer part should be assigned to 10. So, I should write like this; int casting the type casting has to be done and then I should write 10.5. So, in this case I am explicit. So, if I write like this then the compiler knows that what the user wants is the 10.5 value should be taken as an integer value and then assign to x.

So, there are well defined rules by which this transformation will be done. So, it will remove the fractional part and it will take the integer part only and assign it to x, but at this point what the compiler will do or what the what will happen when the program is executed. So, it is not sure. So, in this case it will generate an warning. So, this is an warning that there is a this is the incompatible type assignments. So, when you go to the type management or this type checking part of this course. So, we will see this thing in more detail. So, what you essentially mean is that, it is trying to it is trying to assign some new value to it is trying to assign some new value to a variable, and the value may not be comfortable.

So, that can give some warning. So, there are other types of warnings also that can come. So, many time when you are writing programs and compiling it so, you must have seen many such warnings and most of the time we say ignore the warnings. But ignore the warnings is not always good because as I have said that the value assignment may not be predictable. So, as a result your programmer may not run correctly. So, it is ideally desirable that you remove you take care of the warnings also and take appropriate actions in your program modify the program in such a fashion, that the compiler does not given give the warning also.

See if there are errors in the program, then the compiler does not generate the final target code it gives the errors and come out. But if there are warnings, the compilers they generate the target code also, but the target generated code may not be very good say it may be erroneous. Then primitive system as I said the early computer systems that they did not have compilers. So, at that time the electronic design itself was so, challenging, that people did not think about how to about this writing programs in high level language. They thought that entering the values through some switches and all that should be good enough to judge the underlying hardware. And program written in assembly language and hand coded into machine code.

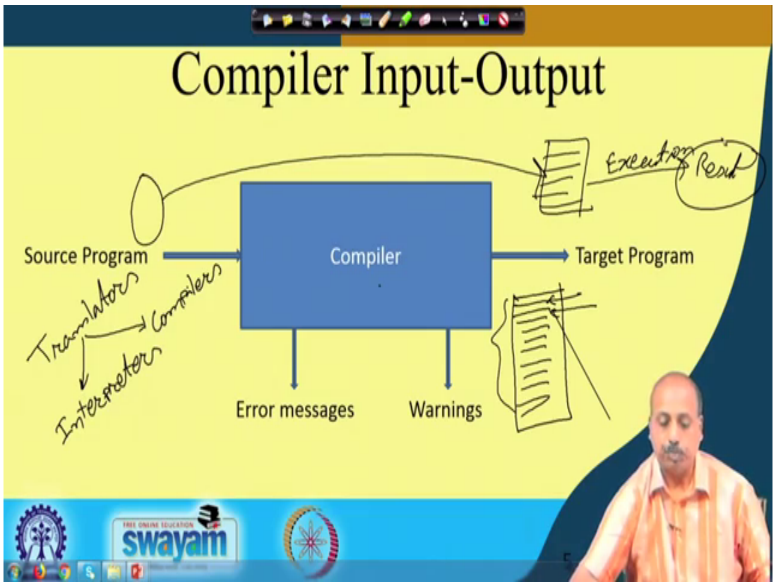
So, after this machine language we have at the next level of program. So, there was a assembly language code and when we discussed some in some courses on microprocessors you might have seen this assembly language programming. So, this assembly language programs so, they are they are they were hand assembled. So, if you look into the processor designers manual, you will find the corresponding code for each and every instruction and accordingly every instruction ultimately gives rise to some hexadecimal code and those hexadecimal code values are entered through manually, through the some card reader or some other mechanism. So, that the program is entered into the computer system.

So, it was very common that ok. So, last night you enter your program. So, you typed punched cards a good say may be a 84 for (Refer Time: 16:30) 89 program. So, there are 80 cards. So, 80 cards you punched on a card punching machine, give it to the system and next day morning you come just to find that, at card number 8 there is a syntax error. So, in and then again you go back again punch the cards correct it, and then again give it. So, that was the problem it was taking huge amount of time today you do not even think about that type of situation. So, as soon as we write the program even a part of it we give it for compilation and see whether there is any error in that part or not. So, that is the beauty of this compilers that can help us in deducting the bugs much earlier.

So, compiler design it started actually with a language FORTRAN in 1950s. So, FORTRAN is the language which is used for which was used for scientific calculations full form is for formula translation. So, and quite sometimes FORTRAN existed and this was also first compiler they were designed for FORTRAN language. And many tools have been developed for compiler design automation. So, as I said that a part of this course. So, they have got foundation on automata theory and accordingly we can come up with some tools automated tools that can generate the compiler given the given the grammar of the language.

So, it can generate the compiler in the sense that it can do syntactic checks ok. So, if there are some grammatical errors in the program. So, it can come up with those errors, but it cannot generate code ok. So, code generation part. So, we have to rely on some other translation mechanism, which are commonly known as syntax directed translation. So, which will be doing this translation part. So, one part of the course is to check whether the program is syntactically correct or not, the second part of the code is to see how we can generate code for the programs which are syntactically correct. So, we will slowly go into this different parts, next we will see.

(Refer Slide Time: 18:49)



Compiler Input-Output

So, pictorially you can think of a compiler like this it us a box. So, that the source program is the input and it the source program is taken as input and it the compiler analyses the source program, and it generates the some error messages if there is some error in the programs. So, it can generate some error message. If there are some warnings so, it can generate warnings and if there are no errors then it also generates the target program.

So, in the maybe in machine code maybe in some other form it also generates the target program. Now at this point of time so, I would like to emphasize that there are the another. So, in general I can see the compilers they are working as the translators. And these translators there are two types of translators that you can see in a computer system; some of them are known as interpreters interpreter and others are known as compilers ok. The role is more or less same in the sense that both of them translates the source program into machine language program, but interpreter is doing it like this. So, if this is a program ok. So, it has got this program lines in it. So, interpreter will take one line at a time. So, it will take the first line, it will convert it into the machine code and it will execute it.

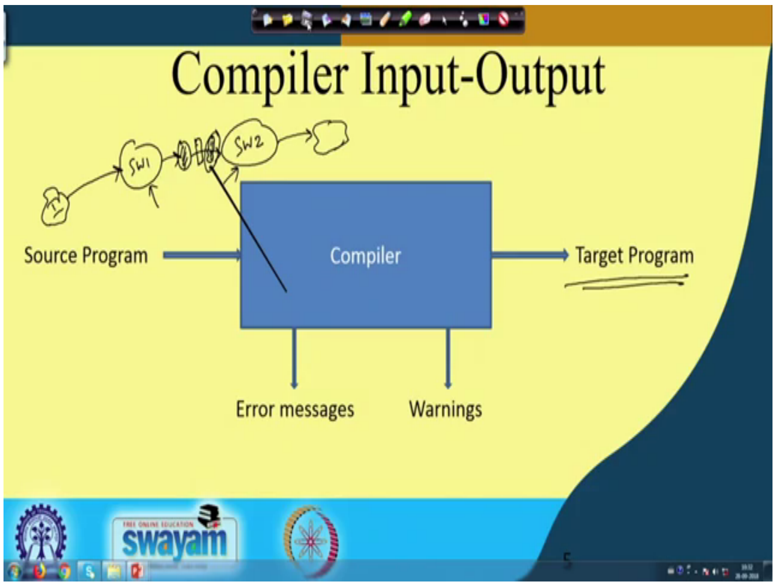
So, if the execution is fine then depending on the execution. So, then it will go to the next statement, it will translate the next statement into machine language and then again it will execute the program. So, it will execute that particular line. So, in case of interpreter as the name suggest. So, it interprets the programming language statements one by the programs statement one by one and executes it. Whereas, for compilers it takes the entire program together and as this diagram is showing this diagram is showing. So, it is generating the corresponding target program. So, given if this is the source language program so, for this it is translated into a machine language program by the compiler. And this entire program can now go into execution. So, it can go into execution and to produce the result it will produce the result. So, that is the works this compiler works.

So, this entire translation is done in one shot. So, this interpreter they are good in the sense that when you are designing the program initially, the interpreters may be good because it will it will go one line by line. So, if there are some error in the program particularly the logical bugs in the program, then it is easy to catch those logical bugs by means of interpreters whatever point there is some problem raised. So, we can we can check the variable values and you can try to see like what has done wrong. But at the same time the interpreted execution is also pretty slow, because it is doing one program line at a time so, it is going to be pretty slow unlike the compiled version.

So, it may be advisable that it initial phase of program development, we use the interpreters and at the final phase when the interpreted programs are running correctly, we are happy with its execution. So, we can give it give the program to a compiler. So, that the compiler will generate the target code directly and this now the target code execution will be much faster compared to the interpreter interpretation of the source language program. So, the interpreter they are working at the source language level whereas, this compilers they are also working at the source language level, but they are converting entire program into machine code. So, as far as execution is concerned it is for the entire piece of code that the execution is being done.

So, this translation so now, we will try to emphasize on this line target program. Now this target program invariably we have assumed so, far that this target program is the machine language program, but it need not be so. So it may be something else also.

(Refer Slide Time: 23:12)



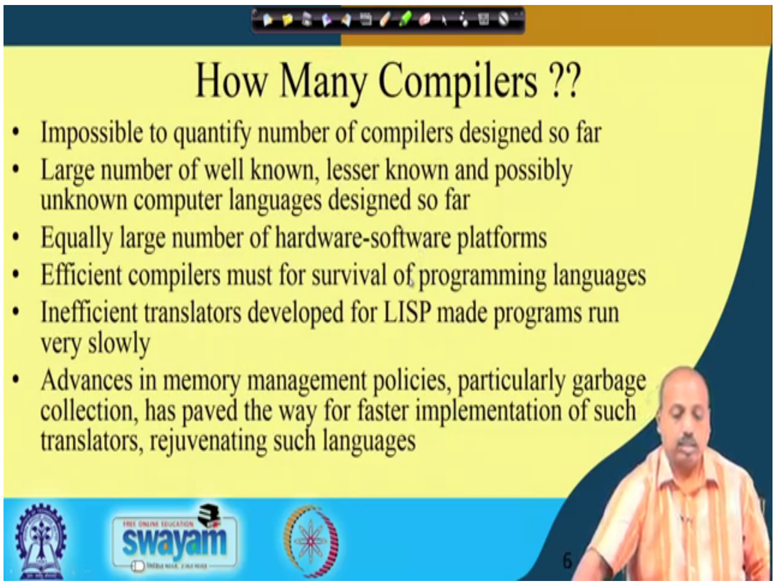
Compiler Input-Output

For example suppose I have got two pieces of software. So, suppose I have got two pieces of software: software 1 and software 2. So, some input that the that comes to the software 1. So, this is the input given to software 1 and it is necessary that this input will be should be processed by software 1 and software 2 to produce the final output. So, this software 1 produces some output that goes as input to the software 2 and the software 2 finally, produces the output. So, it can. So, happen like this.

Now, if this software 1 and software 2 are coming from two different vendors. So, it is very much possible that this format in which this software 1 produces the output is not compatible with the input format of software 2. So, in that case it is necessary that we put a translator here also. So, that this program is this output of software 1 is converted into another output, which is compatible to software 2 which is understandable by software 2.

So, in this case this target program that we are talking about is this one this is a second program that we are these input or software 2. So, here also this compiler that we have so, it is doing a translation, but it is generating target program which is not for machine execution, but for understanding of another piece of software. So, that can happen. So, that is why this target program when you say. So, it need not necessarily be the machine language program it maybe some other format also. So, whenever we need any format conversion type of job. So, this compilers so, they can be utilized.

(Refer Slide Time: 24:59)



How Many Compilers ??

- Impossible to quantify number of compilers designed so far
- Large number of well known, lesser known and possibly unknown computer languages designed so far
- Equally large number of hardware-software platforms
- Efficient compilers must for survival of programming languages
- Inefficient translators developed for LISP made programs run very slowly
- Advances in memory management policies, particularly garbage collection, has paved the way for faster implementation of such translators, rejuvenating such languages

6

So, next we will see. So, what are the next answer very simple question like, how many compilers? Like if we say that how many compilers have been designed so, far can we enumerate and so, it this slide is just to make you understand that how difficult maybe the job of a compiler design ok. So, impossible to quantify number of compilers designed so, far. So, many compilers have been designed. So, nobody can tell like how many compilers have been designed so, far.

Large number of well-known, lesser known and possibly un known computer languages designed so, far. So, after we have gone through this code. So, you will understand that designing a new language is not that difficult. So, once you are very clear in your mind that what are the requirements that I need to specify, you will see that designing a new language is not difficult and whenever you design a language. So, you need to have a compiler to translate it into some other form. So, it may be machine code may be something else, but we have to do that.

So, as a result what has happened is that, various people they have tried to design new languages and accordingly there are lots of languages that have been designed. So, which are lesser known and possibly unknown computer languages. So, need not known at all and similarly there are large number of hardware software platforms ok. So, hardware software platforms so, every day some new company they are coming up with new features they are adding new features to the operating system. So, if the compiler is trying to exploit those features, then definitely it has to be it is a new compiler because it is targeting to a new target machine.

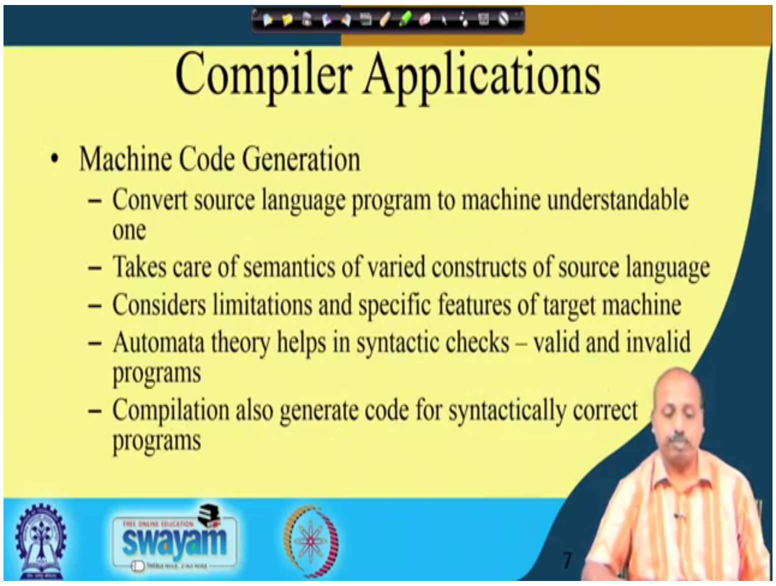
So, that way this how many compilers if I want to answer. So, it is very difficult to answer in terms of numbers possibly you can say it is infinite. So, efficient compilers are must for survival of programming languages, because if the compilers are not efficient then nobody will be using that particular programming language. For example if I design a language say l and I find that whenever I write a programs in language l. So, when it is translated into machine code, the program is much much slower than the programs that we have in C language. So, in that case the language l will not survive people will not use this language l ok.

So, that way these efficient compilers are very much necessary for this programming languages to survive; inefficient translators developed for LISP made programs run very slowly. So, LISP is a functional programming language. So, that is that was designed long back ok, but in it is it has got very nice features like programs that you write in LISP are very much much smaller compared to the high level language programs other high level language program like C and all, but the problem is the translated version of the program. So, they were very slow.

So, if the translators designing efficient translator became a problem was a problem. So, the lisp program they use to run very slowly. So, it was not that much popular. However, with the advancement in memory management policies particularly garbage collection and all so, it has provided the avenue by which you can have a faster implementation of such translator and such languages are rejuvenated.

So, that way many old languages they are also coming back and because of the facilities that are provided in the operating system and the underlying hardware, and accordingly the programs they are they can be the efficient can be generated for the languages. So, this way answering the question how many compilers have been designed so, far is not at all very straight forward, you can only say it is not enumerable.

(Refer Slide Time: 28:55)



Compiler Applications

- Machine Code Generation
  - Convert source language program to machine understandable one
  - Takes care of semantics of varied constructs of source language
  - Considers limitations and specific features of target machine
  - Automata theory helps in syntactic checks – valid and invalid programs
  - Compilation also generate code for syntactically correct programs

7

Applications of compilers; so, you have got first application is the machine code generation. So, it convert source language program to machine understandable one. So, that is the first the very first thing, that most compilers they are designed for this machine code generation takes care of semantics of various constructs of the source language. So, this is; obviously, I have to see that the program that is given to given for compilation is syntactically correct. So, that is the syntactic the semantic constructs and syntactical structures that we have so, that they are satisfied or not.

Then limitations and specific features or target machine. So, it has to see like what are the features like for example, say integers in the in some machine is allocated say 2 bytes some machines 4 bytes in some machine it is a little Indian architecture some machine it is a big Indian architecture. So, like that there are different architectural features that we have. So, I have to look into those limitations and features of this target machine to see how the compiler should generate the code.

Automata theory that helps in the syntactic check checks; so, it can classify the program to be either a valid one or an invalid one, but it does not answer anything beyond that yes and no. But if the answer is no in that case also the compiler has to give enough indication that what exactly went wrong in the program ok. So, that is the challenge. So, it is not that compiler design the compiler designer designs compiler such that given a wrong program, it just says that the program is syntactically wrong should not be. It should tell at line number 10 there is this error, at line number 15 there is this error, line number 30 there is this error.

So, you see that we get this type of messages. So, how this is actually done whereas, the basic theory of automata so, that will allow us to classify the programs only into two classes valid program and invalid problem. So, it does not give us hint on where exactly it went wrong.

So, it is the compiler designers responsibility to see that to see that enough information is given back to the programmer, to indicate how much where exactly the program went wrong and what what should be the action. Compilation also generates code for syntactically correct program. So, if the code if program is grammatically correct then the compiler should generate the code. So, this is the major operations that that are done by a compiler in the machine code generation process.