

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

Lecture – 02
Introduction (Contd.)

(Refer Slide Time: 00:15)

Compiler Applications (Contd.)

- Format Converters
 - Act as interfaces between two or more software packages
 - Compatibility of input-output formats between tools coming from different vendors
 - Also used to convert heavily used programs written in some older languages (like COBOL) to newer languages (like C/C++)

The slide includes a diagram showing a flow from an input box to a box labeled 'T1', which then connects to a box labeled 'T2'. There are also some handwritten annotations and a small video inset of the professor in the bottom right corner of the slide.

Another application that we have for compare for this compilers, are the format conversion. So many times what happens is that we need to, as I was telling that we have got 2 different software packages, and between the software packages the output of one software package should go to the as a go as input of another package. So, accordingly we need to give some translation. Because the, there may be the input output formats between the tools may not be compatible, I mean, particularly if they are coming from different vendors.

So, in that case, so the software output that we have from one tool. So, if this is say tool 1. It takes some input and it produces some output, and then we have got tool 2, but this tool one output I need to feed it to tool 2, but it is not compatible. So, tool 2 it expects input in some format. So, it is very much common particularly like maybe in tool 2 the input format. So, it does not take semicolons at the end of the lines whereas, tool 1 it produces semicolons at the end. So, like that so maybe tool 2 it requires the variables to be defined in some fashion whereas, tool 1 output is in some different format.

So, in those cases what is needed is that in between we put a translators. So, this is a another compiler, and this translator produces the output which is understandable by tool 2, so and it goes like this. So, this way we have some format converters. And this is very much common, particularly if you look into the CAD domain, VLSI, CAD domain. So, you will find the many such tools where we need to do lot of format conversions. So, this is compatibility between input output formats between tools of different vendors.

Now, next thing is that also it is used for converting heavily used programs written in some older languages; like, COBOL to newer languages like C C + +. So, COBOL is a language common business oriented language. So, that was used at one point of time; so they were used very much, particularly for these office automation purposes where we used to generate this payroll and all using this COBOL programs.

Now though this COBOL as a language is very much (Refer Time: 02:42), and it is the programs that are that runs in COBOL. So, they are not very much efficient as far as execution speed is concerned, but they are very efficient as far as file handling is concerned. On the other hand, today after that this language is like C and C + + came, so now the new software they are be that are being developed.

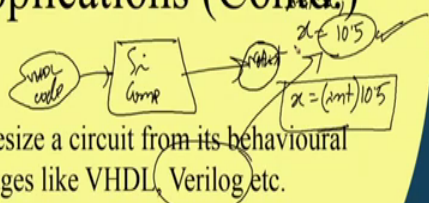
So, they are not they are not being done in COBOL, but in some next level programming languages or even in 4 generation language. But that point is there; so there are large number of programs which were existing. So, this payroll software that we had so, that is that was existing in some old language like COBOL. So, we need to convert we do not want to make though all those programs junk, rather we try to have some converter so that we can automatically generate the C C + + programs from those old COBOL programs. So, that way we have got, we have we have to have this programs which are written in some older languages translated into newer languages.

So, that is the role of format converter, so compare, so here also we have got the compilers in picture.

(Refer Slide Time: 03:57)

Compiler Applications (Contd.)

- Silicon Compilation
 - Automatically synthesize a circuit from its behavioural description in languages like VHDL, Verilog etc.
 - Complexity of circuits increasing with reduced time-to-market
 - Optimization criteria for silicon compilation are area, power, delay, etc.



The diagram illustrates the silicon compilation process. It starts with a box labeled 'VHDL code' with an arrow pointing to a box labeled 'Syn Compiler'. From the 'Syn Compiler' box, an arrow points to a box labeled 'Circuit'. To the right of the 'Circuit' box, there are handwritten notes: $x = 10^5$ and $x = (int) 10^5$.

Next we have got another application which is known as silicon compilation. So, silicon compilation is like this; that today suppose we are trying to design some new integrated circuit chip. For example, suppose we want to generate a new processor. Now if we want to design a processor, now it if there are there can be 2 different avenues by which we do it. In one avenue, we start with the lowest level modules like, the lowest level digital circuits like adder, subtractor, multiplier, register design, then slowly go up and go up to the full system design.

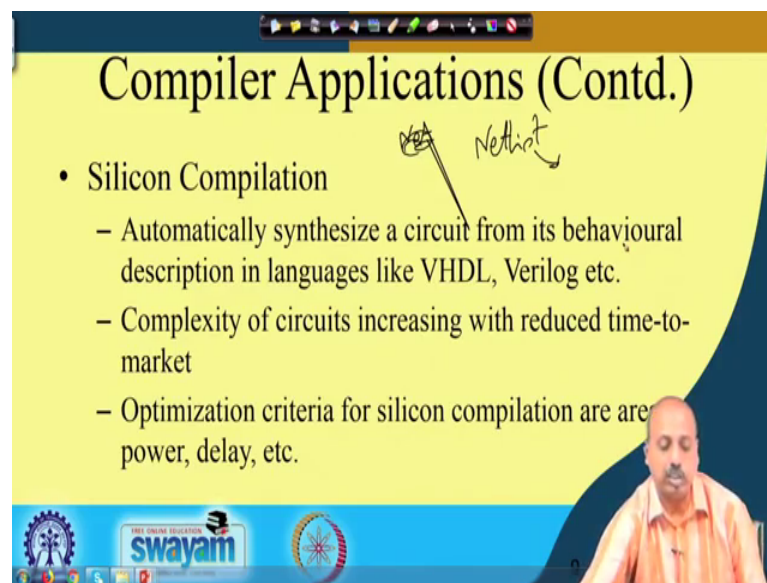
So, but the problem in this particular approach is that it is time consuming, and if the user is not very much careful. So, there is a possibility that some bug will get introduced into the process, and then the chip that is fabricated is not correct. On the other hand, so there can be another avenue by which we can handle this problem is start with the behavioral description of the system in some language hardware description language which is like VHDL, Verilog. So, these are popular hardware description language in which you can describe the hardware in different levels; so, behavioral level, structural level, functional level like that.

So, at the top level we have got the behavioral level description. So, you can describe the behavior of the processor in terms of this language constructs. And then from they are we try to generate the circuit. Now you see here the input to the system is a description in some VHDL or a Verilog language. And then the output that we have, so if this is the

silicon compiler, if this is the silicon compiler, so here you have got this VHDL code as input and as an output, you do not have machine code, but rather you have rather you have got some other you have got another VHDL code which is basically a structural level description which we call netlist, which we call netlist.

So, this is the netlist of this library components that we have.

(Refer Slide Time: 06:21)



Compiler Applications (Contd.)

- Silicon Compilation
 - Automatically synthesize a circuit from its behavioural description in languages like VHDL, Verilog etc.
 - Complexity of circuits increasing with reduced time-to-market
 - Optimization criteria for silicon compilation are power, delay, etc.

Netlist

So, that netlist sorry so, net that netlist that we are talking about. So, this netlist is nothing but it is a collection of hardware modules that will be making that will be making this whole system. For example, in your system you will need adder, subtractor, registers etcetera. And some connection pattern between them. So, this netlist will be a connection it will having all those module descriptions and it will it will have the connections between them. So, this thing is automatically synthesized. So, user does not design the individual adders. So, they are designed previously and available in the library and it is taken from there. So, that way it is done. So, for very complex systems, so it is so this is the way that we should proceed, we should know it should start with the behavioral description. And then convert it into the netlist it depending upon some library modules.

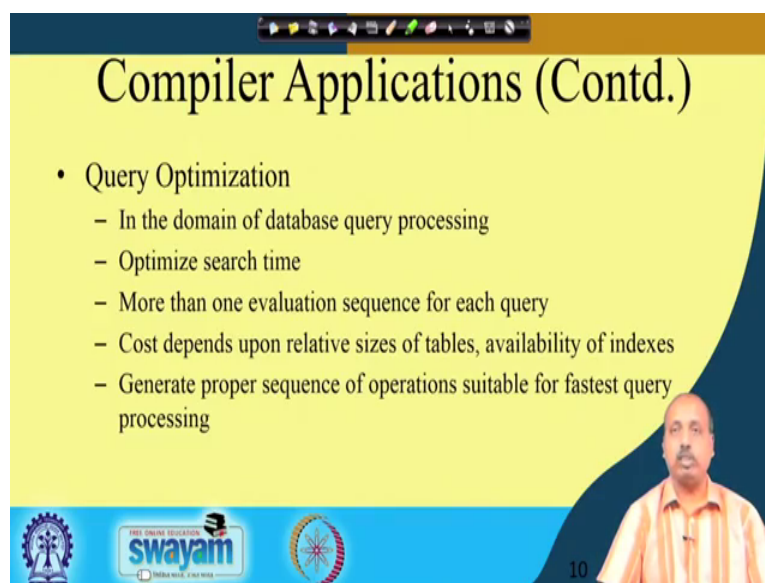
So, this is, this is known as the silicon compilation process, and this is also some sort of compiler. So, complexity of circuits increasing with reduced time to market. So, thus so this is the pressure that we have. So, 2 motive every day we are coming up with newer

and newer systems electronic systems that have got more and more functionality into it, and the and that there is a very high pressure on time to market. Like, if a if you if we survey the market and see that we want to introduce a new cell phone into the market, then maybe if it is introduced with the next 6 months there will be some benefit some profit, but if it is introduced after one year maybe that profit will be half, and if we introduce after say one and half year maybe the profit will become 0.

So, that way there is a very high pressure on this really on reducing this time to market. So, here if you are starting at the lowest level and trying to generate the whole system very efficiently so it will take time. So, we go in the other way, so we start with the high level description behavioral description of the system and from there try to come to the circuit. And the optimization criteria they are also different. Like when you are writing programs for execution by a processor, then the optimization criteria there is the speed of execution, then you have got this what is the memory requirement. So, these are actually the optimization criteria.

But when you are doing silicon compilation, so the optimization criteria is the overall area required by the circuit, the power consumption of the circuit the delay that the circuit will have. So, we want to reduce this area power and delay. So, the measures that we have are totally different. So, a compiler which is targeted towards say machine code generation we will not do well, when if you put it onto a silicon composition process. So, the challenges are totally different.

(Refer Slide Time: 09:25)



Compiler Applications (Contd.)

- Query Optimization
 - In the domain of database query processing
 - Optimize search time
 - More than one evaluation sequence for each query
 - Cost depends upon relative sizes of tables, availability of indexes
 - Generate proper sequence of operations suitable for fastest query processing

10

Next, the other application that we have is in the domain of database query processing which is known as query optimization.

So, in database you know that a major operation that we have is to answer the user queries. And once the database has been created, its content is more or less static. For example, if you look into the database of an organization in the employee database of an organization. So, initially there will be additions and all, but after sometime after the database has been stabilized. So, it is not that everyday's a lots of employees are joining the companies and leaving the company. So, that way that database is more or less stabilized or more or less static in nature. And the queries that you get on the database are like this maybe it is trying to find out the average salary, or it every month we have to do salary processing and all that.

So, the queries are much higher in terms of some information taking getting some information from the database, but the basic data value changes are much less, ok. So, it is very much important that whenever we have got this query. So, these queries are to be processed, and they are to be answered quite fast. Now how to reduce this search time? So, if we look into the database theory you will find that there are depending upon the size of the tables that we handle.

So, we can handle we can manipulate the tables in certain order to reduce the overall time requirement. So, given a query what is the exact sequence of operations that we should

do on the database for finding the answers quickly? So, that is a big challenge, and that is known as the process of query optimization. So, that is also some sort of compiler. Because you see, that the given the query in some high level language like say sql and all, so converting it into some file operations. So, ultimately you are it is it is doing some file operations over the ultimately the operating system file routines are to be called.

So, how this translation will be done? So, that is a big question and we have got these compilers to help us. As it is noted here that we want to optimize the search time; so, more than one evaluation sequence may be there for each query, and the cost depends on upon the relative sizes of tables availability of indexes etcetera. So, the same query if the table sizes are different. So, it may be if you process in different sequence of operations, then the timing requirement will be different. So, as a, so we definitely do something. So, that a particular ordering has to be found. So, the size of the tables will dictate like other with the sequence in which the query should be evaluated. So, generate proper sequence of operations suitable for the fastest query processing, so that is very important.

So, here the optimization is not in terms of. So, the space requirement or that execution time, but it is in terms of the retrieval from the database. Like if you know that these are these are file operations. So, whenever you have got file operations involved. So, that file operation take very high amount of time compared to the processing. Like once the ones are an employee salary record has been retrieved; so, maybe we just want to increase the basic pay by some amount.

So, that operation is just an addition operation or at most another multiplication operation. But the major time is spent in getting the record from the secondary storage. So, the optimization that we have is in terms of the number of disk accesses and all. So, the optimization criteria is totally different when we are going for going for database operation, compared to the situation that we have for say, this may code like say execution of normal programs, ok.

(Refer Slide Time: 13:15)

Compiler Applications (Contd.)

- Text Formatting
 - Accepts an ordinary text file as input having formatting commands embedded
 - Generates formatted text
 - Example *troff*, *nroff*, *LaTeX* etc.

Handwritten notes: 'Graphical' with an arrow pointing to a box labeled 'formatting commands' and 'text'. A small video inset shows a man speaking.

Another very interesting application that we have for compilers is in the text formatting. So, there are many tools by which you can format the text files. Like, today you know that there are many such formatting tools. For example, we have some of these thumbs, so these formatting tools that we have. So, they can be classified into 2 categories, sorry. So, this formatting tools that we have so this text formatting some of these tools. So, they are what we are doing? So, we are providing some we have provided some sort of graphical interface, and in that graphical interface the font and everything is designed, and we just choose the appropriate font and enter the text in that format. So, immediately you get a feedback, like you are writing in such and such font and font size, and also you immediately get a and get an understanding like how is it looking like.

So, you can you can choose some other format, you can make some part bold and all by doing it immediately on to the text. In some other format, so the it is like this that if you just write a file, and in that file you put your normal text, and before this takes you put some and also embed some formatting command in it, some formatting command you input into it in the same file.

So, in this case what will happen is that, so now, this whole thing is given to a compiler and the compiler; so it applies these formatting commands on this text, and then it will be converted into a formatted text. So, here in this in case of graphical thing what is what was happening is, it was that you are just you are immediately looking getting this

formatted text and as a visual output, so you are just looking into that. But in case of this one; so this file that we have is nothing but a simple text file. And this simple text file is converted by means of a compiler into a formatted text file where this formatting commands are taken care of. Like this formatting command maybe it will tell that the next few lines, I want to make it alex. So, accordingly it will do that.

So, it has got many advantages, because this they have the portability of this file is very easy. So, because this is a simple text file so, you can very easily transfer and all, and you can, so after some after some practice and experience. So, you will you may find that these formatting tools are much better compared to this graphics based tools, ok. So, this text formatting tools, so they come under this compiler. So, this is also some sort of compiler, because you are accepting input in some text file format and then as an output you are producing this graphical thing, so that is also a text formatting. So, this text formatting tools. So, they accept ordinary text file as input, having formatting commands embedded into it and it generates the formatted text.

And. So, these are some of the example like, so in Unix operating system you have got troff, troff, nroff and there is another very well-known package which is known as LaTeX, ok. So, the where which are used for doing this type of translation so; they come under the broad heading of formatting tools. So, that is also a type of compilers.

(Refer Slide Time: 16:51)

Phases of a Compiler

- Conceptually divided into a number of phases
 - Lexical Analysis
 - Syntax Analysis
 - Semantic Analysis
 - Intermediate Code Generation
 - Target Code Generation
 - Code Optimization
 - Symbol Table Management
 - Error Handling and Recovery

Handwritten notes on the slide:

- English \equiv Alphabet = A, B, ..., Z, a, b, c, ..., z, 0, 1, ~, 9, ...
- Does not exist any hard demarcation between the modules.
- Work in hand-in-hand interspersed manner
- Sentence collection of word

The slide also features a logo for 'swayam' and a small image of a person in the bottom right corner.

Next we will look into the phases of a compiler. So, you see that if I have got a big job to do like this called translating from one language to another language, maybe high level say C language to target machine code, or say some a VHDL to this silicon output, or say this formatting tools text formatting tools, or whatever for whatever the application we think about.

So, this transformation process is quite complex. So, ideally I can have I have a monolithic piece of software; which is doing this translation the compiler appears to be monolithic, but in the design phase just to make this compiler design process simpler. So, we can we can think about it to be divided into a number of phases, though practically speaking, so there is no hard demarcation ok. So, these modules are they are they are not they are not demarcable clearly. But for our understanding for our discussion in the course, so we will be dividing them into number of phases ok. So, the first phase is known as the lexical analysis phase. So, to start with, so before going into this, so let us try to understand; like, if I have got some language, for example, if I have got the language English ok.

So, any language, so what we have immediately is the first thing that we have is the alphabet. Any language starts with the alphabet. Now you know that in English language the alphabet set is say this capital A, B, etcetera up to capital Z, then this small a, small b, small c, etcetera up to small z. Now there are many more symbols like this, 0, 1, to 9. Then whatever special symbols that we are allowing in say today's English language text. So, they all come under this alphabet set. So, for any language it starts with the alphabet ok. So, we have we have we are having the alphabet. Now once I know the alphabet set. So, this is commonly represented by the symbol sigma ok. And then this alphabet some of these symbols in the alphabet they are combined to form words.

So, this alphabet is from the alphabet we get words. Word is nothing but collection of alphabets, and it is separated by some special say separator for example, in most commonly we are using the separator blank ok. For English language sentence, so this word so word, so any collection of alphabet you take ok. So, that is a word now this word may be a valid word may be an invalid word. So, word you can classify into 2 categories one is the valid word and another is the invalid word. For example, for English language c a t cat, so this is a valid word as far as we know. But say c t a this

possibly not a valid word. So, I am not very much knowledgeable in English language that way.

So, I cannot say very clearly that whether cta is a valid word of English or not, but assuming that it is not a valid word. So, it goes into the invalid category. So, that is the second thing that we have. So, alphabet said, so if there is a, so if my English language does not allow the Greek symbols like alpha beta etcetera, then if I get the symbol beta somewhere in the text, then I will say this beta is not in the alphabet set. So, there is a problem with the alphabet, but even if the alphabet part is correct, this cta is not a valid word. So, that is another level of invalidity. The next level of invalidity that comes like when we consider the sentences; when we consider a sentence now sentence, so it is a collection of word collection of word.

Now, if we consult the grammar of this English language grammar of the English language. So, there are certain rules which will tell us like what are the valid sentences what are invalid sentences and all. So, it is a very huge process or I should say very complex process to say whether a language whether a statement that I have made in English language whether it is grammatically correct or not. But anyway, so if we take some simpler language maybe the grammar rules will be simple, and it will be very it will be easy to tell given a sentence whether it is whether it is a whether it is valid for the language or not. So, accordingly you can say that this language this sentence it can also be valid sentence or it can be invalid sentence. Now this if a sentence is valid sentence, we say that if the sentence belongs to the language. If the sentence is an invalid one, we say that it is not belonging to the language.

So, that way we can have this foreign language starts with alphabet, then goes to words, then goes to sentence, at each level we have got the issue of validity and invalidity. So, in the lexical analysis phase so, what we will try to do is to see whether we have got some valid word or not. So, alphabet we can check immediately. So, whether there is any foreign character that are present in the in the description, and if there is there is no if there is no foreign character.

So, it will try to see like what are the words of the language that are appearing in the disk in the program. So, accordingly it can determine the words. So, it can it will do the processing and it will find out the words. So, that is the job of lexical analysis. Once this

lexical analysis is done, we have got the phase of syntax analysis. So, in the syntax analysis phase we do the grammatical check, then we have got semantic analysis, where we try to find out whether there is any problem with the meaning like is maybe some variable is undefined and all that.

So, that is a semantic analysis phase. Then we have got some intermediate code generation phase so, up to this semantic analysis phase. So, you can or the syntax analysis phase, you can say, so this is based on automata theory. After that whatever we will have, so they are more they will be using the cost of the outcomes of this automata theory based analysis, but they will be augmented significantly by means of other techniques so that we can do the code generation part. The first phases of that is intermediate code generation. Then that intermediate code is translated into target code, so that is the target code generation. Then after the code has been generated, so we can stop there, but most of the compilers they will go into a code optimization phase.

So, since this is the entire process is an automated process. So, there is a very high chance that the code that is generated by this code generation process is not optimized significantly. So, it will be optimizing it, and then maybe we will be getting a faster code for that. So, that is the code optimization phase. So, apart from that, there are some other points which are also important for compiler design. One is the symbol table management. So, symbol table is a table of all the symbols that appear in the program. All the variables, procedures labels, that appear in the program. So, they come under the symbol table part. So, the symbol table has to be managed, because at many times the compiler module, so they will refer to this symbol table for the checking and code generation process, so they will do that.

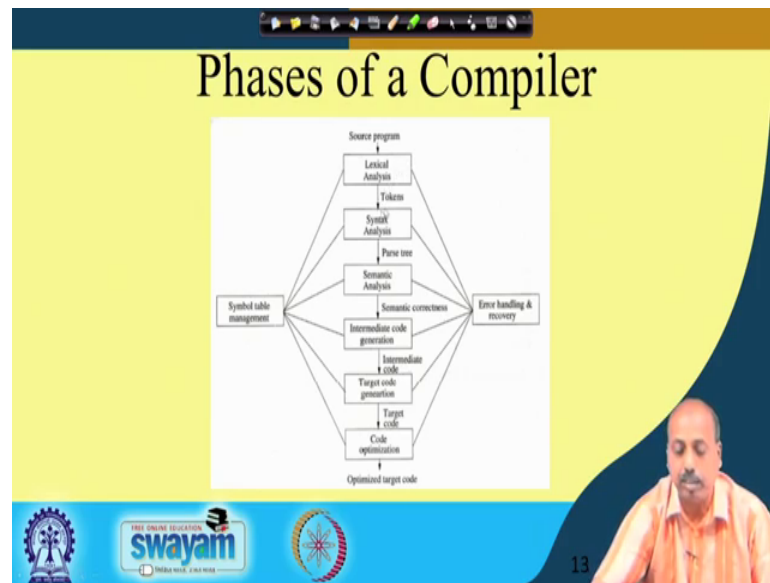
And we have got another very important phase which is known as error handling and recovery. So, whenever there is some error in the source language program, then this compiler has to detect it, and after detecting it has to proceed like, if there is in there is an error at line number 10, then it is not advisable that the compiler just prints that there is an error in line number 10 and comes out. Rather it should check the remaining part of the program also, and come up with a list of all those lines. So, where the compiler things that there is an error maybe there are errors at line 10, 15, 18, 19, 20 like that. So, all those lines as many as possible so, if we can list it, then in the next go

the programmer may correct all those errors and then come up with the corrected version of the program.

So, that way the program works life will be simpler. Otherwise the programmer corrects line number 10 give you gives it for compilation and finds that again the compiler has given an error, that there is an error in guideline number 15, so that is not very much advisable. So, this error handling and recovery, so this is important so, the recovery part will be clear when you go into the details of this error analysis job ok. So, it is not very much understandable now. So, there is a caution, of course, that you should not think that all these modules are totally dealing from each other. It is not that the output of one module is given entirely the second module then the second module starts, so it is not like that. So, there is not much demarcation between the modules, and they work hand in hand interspersed when.

So, it is like this that whenever syntax analysis phase, it will find it may find that it needs some more words next word from the system from the from the program. So, it will ask the lexical analysis phase to give it that next word. Similarly, when the code generation phase, so it will try to generate the code, it may ask the syntax energy analysis phase to generate the to give it to tell it a what is the next rule by which I should proceed what is the next grammar rule by which I should proceed. So, that way all these modules they go hand in hand. The later part like code optimization and also they are a bit independent, but up to this intermediate code generation phase. So, I should say that they are all interlinked with each other.

(Refer Slide Time: 27:27)



So, next we have this diagram actually depicts the whole thing. Like, your source language program it enters into the lexical analysis phase. Now, lexical analysis phase, so it does many thing. Like, there are there normally the pro source language program they have got a lot of comments. So, the comments are early move, because comments are never translated into machine code. So, they are removed from the program. Then between 2 words somebody may put one blank somebody may put 10 blanks like that.

So, all those extraneous blank spaces they will be removed. And many times we have got some header files, we include some header files, and we say that the compiler will for example, in C language we have got the hash include directive, so they are all expanded ok. So, this is all this is expansion will be done by the lexical analysis phase. So, ultimately this lexical analysis phase, so it will generate some tokens. So, this tokens will be used by syntax analysis phase. Syntax analysis phase generates parts tree which is used for semantic analysis. That is, the is the program is semantically correct, then it goes into the intermediate code generation phase. Then from there the intermediate code it is translated into target code, goes to target code generation phase. And from the target code it goes into optimization phase.

And the symbol table management it is used by all the modules, from lexical analysis to code optimization. Some of the phases they will put entries into the symbol table, some of the phases. So, they will use the values that are present in the symbol table. And there

are error handling and recovery routines. So, which will be which are integrated with all these phases again, and if there is some error that has occurred. So, it will be trying to get trying to get some information from different phases and flash appropriate error message to the user so that the user can correct the program in a better fashion. So, these are the different phases. So, by which the compiler generates that I optimized target code from the source language program.