

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

Lecture – 05
Introduction (Contd.)

(Refer Slide Time: 00:15)



The slide is titled "Challenges in Compiler Design" and lists the following challenges:

- Language semantics
- Hardware platform
- Operating system and system software
- Error handling
- Aid in debugging
- Optimization
- Runtime environment
- Speed of compilation

The slide also features a small video inset of Prof. Santanu Chattopadhyay in the bottom right corner. At the bottom of the slide, there are logos for IIT Kharagpur, Swayam, and the Ministry of Education, Government of India, along with the slide number 23.

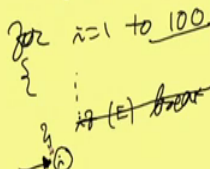
We are started discussing on Challenges in Compiler Design. So, there are several challenges, starting with the language semantics to hardware platform, then operating system concerns. Then the error handling, the powerful error handling will help in the will make the compiler more attractive. Then the debugging aids that is if I want to rectify the logical bugs in the program, then how can I do that, how the compiler can help in doing it.

Optimization; so I would definitely want more and more optimization, but that makes the compilation process more and more difficult; then the runtime environment management, so depending upon their facilities that we want to provide in the source language program like whether it supports, recursion and all; this runtime environment management will vary. And the speed of compilation that is how fast is the compiler, so it should we should not take lot of time for doing the compilation.

(Refer Slide Time: 01:09)

Language Semantics

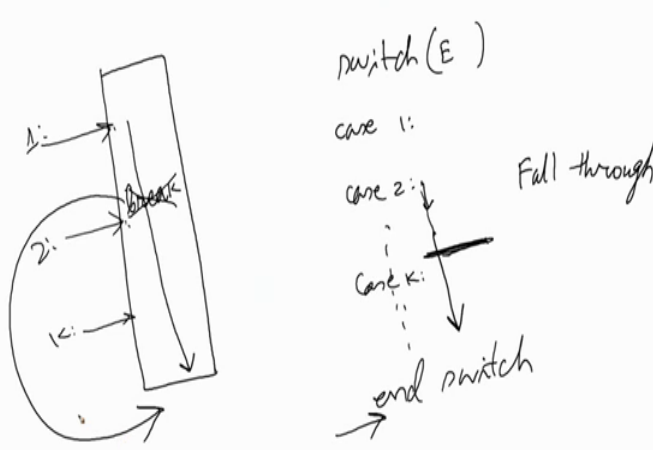
- “case” – fall through or not
- “loop” – index may or may not remember last value
- “break” and “next” modify execution sequence of the program



The diagram shows a handwritten loop structure. It starts with 'for i=1 to 100', followed by a block of code, and then a 'break' statement. An arrow points from the 'break' statement to a circle containing the number 100, indicating the loop's termination point.

So, if you look into the language semantics, so here are a few cases, so which I would like to point out. One is that fall through case statement in most of the programming languages, they will support some sort of case statement. And this case statement in some programming languages, it is a fall through case, and sometimes it is not so.

(Refer Slide Time: 01:39)



The left diagram shows a vertical rectangle representing a loop body. Arrows labeled '1:', '2:', and 'k:' point to different sections of the rectangle. A curved arrow at the bottom indicates a loop back to the start. A 'break' statement is written inside the rectangle, with an arrow pointing to the end of the loop.

The right diagram shows a handwritten switch statement:

```
switch(E)
case 1:
case 2:
...
case k:
    Fall through
    ↓
end switch
```

An arrow points from the 'end switch' statement to the right.

So, we can I can take an example, like most of the programming languages will have some sort of say switch statement switch or case. And some expression, some variable or

expression can be there, some expression E. And then we have got different cases. So, case 1, case 2, so like that we have got different cases.

Now, some programming languages they say that depending upon the value of E. So, it can enter into one of these cases, so this is a case K like that. Now, if E is E evaluates to two, so it will start executing from this point. So, what happens, when it reaches the end of that particular case? So, we can have two types of option. In one option, it will keep the rest of the cases, and it will the end the switch statement ends at this point. Then after finishing this after finishing this line, so it will start executing from this point.

In some other programming languages, so this is called a fall through case. In which ones this ones this case matches value matches, then it execute that part and it continuous so remaining cases are also executed from that point onward. So, it is just the entry point. So, it is like this as if this all these cases, they form a piece of code a chunk of code, and you have got some entry points only. So, this may be the entry point for a 1, this may be the entry point for 2, this may be the entry point for K like that.

So, after you have entered, so this entire piece of code will be executed from that point onwards, until and unless there is a break statement in between, which will be taking it out of the case, so but this is true for languages like say C. But, in some other programming languages, so it is like this so this break is not there, but as soon as this is over.

So, instead of continuing with the next case, so it will come out of the switch block all together. So, these are the two types of case statements that we can have been different programming languages if you look into different programming languages, you can find them. So, naturally the code that we generate for the two cases, so they are different two situation or two programming languages they are going to be different.

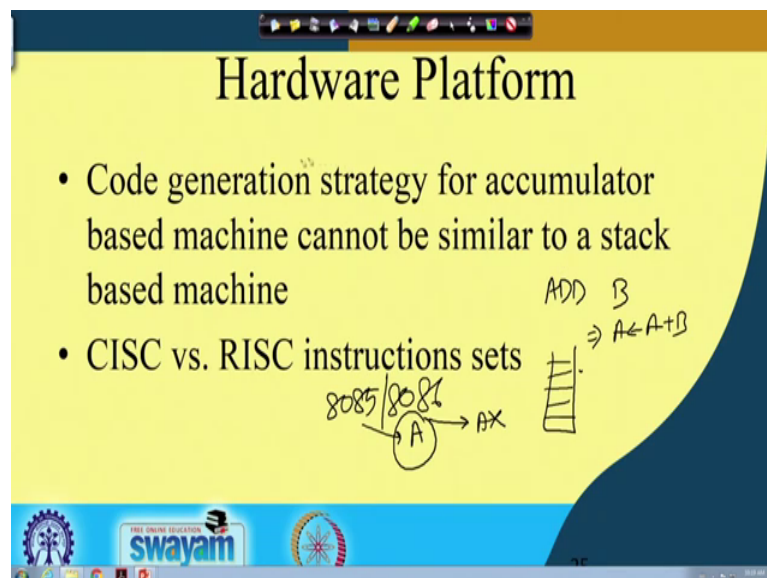
So, another concerned that we have is with say loop statement. So, loop statement, in some cases suppose I have got a loop like this. Say for i equal to 1 to 100, and in the body of the loop somewhere I have a break statement. So, suppose here there is a if some condition is satisfied, then we break from the loop. Then what is the value of i, when you come to this point ok.

So, there can be different types of semantics different type of logic about the value of i . Some programming languages say that when you are coming out with the way from somewhere in between without completing all the 100 iteration, so here you get the value of i , which it had at that point of time. In some other programming languages, it says that no i will be value of i is unknown. So, value of i may not be the value of i may not be predictable, it can have any arbitrary value.

So, this loop index. So, if the value may or may not be remembered beyond the last value or even if this finishes properly, i will become i it goes from 1 to 100, there is no break statement. Then what is the value of i , after the loop as completed. In some programming languages say, it will say that the value of i will be 101. Some programming languages, we will tell that it is unpredictable. So, depending upon that the code has to be generated right way.

Similarly, this break and next statement they modify execution sequence of the program. So, we have to take care of them. So, this way the language semantics they have got many they put many compulsion or many conditions on the code that we are going to generate.

(Refer Slide Time: 05:57)



The slide is titled "Hardware Platform" and contains the following content:

- Code generation strategy for accumulator based machine cannot be similar to a stack based machine
- CISC vs. RISC instructions sets

Handwritten diagrams and notes on the slide include:

- A diagram showing "8085/8086" with a circle containing "A" and an arrow pointing to "AX".
- A diagram of a stack represented as a vertical ladder.
- Handwritten text: "ADD B" and " $\Rightarrow A \leftarrow A+B$ ".

The slide also features logos for "swayam" and "MHRD" at the bottom.

So, the next concern that we have is about hardware platform. So, code generation strategy for accumulator may base machine cannot be similar to a stack based machine. So, in an accumulator based machine, what happens is that there is a special register,

which is called accumulator. And all operations that we do one of the operands and the destination is the accumulator.

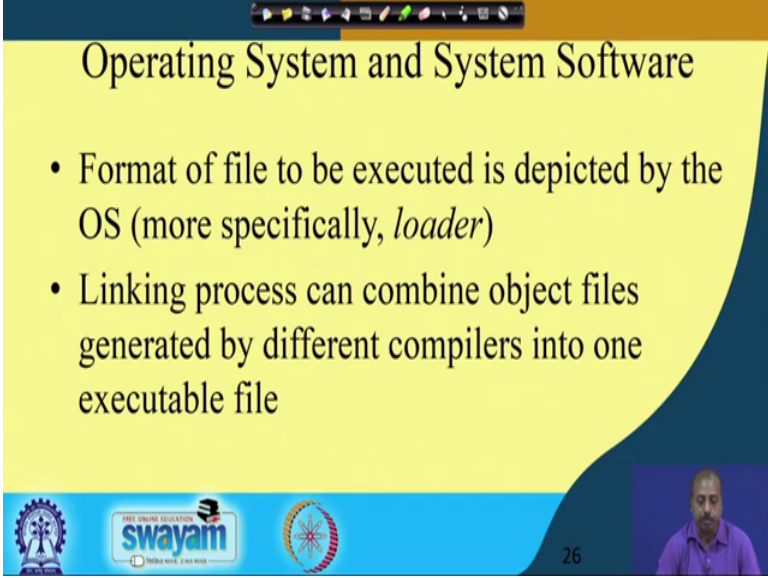
For example, if you look into say 8085 type of architecture or 8086 type of architecture, all these architecture so they have got one accumulator register A in 8085, and in 8086 we have got AX. So, there any operation that you do so particularly for in 8085 architecture, so which simply say ADD B. So, ADD B means, the operation that is carried out is A getting A plus B, so that type of code that we that we generate, so it should be of this nature.

On the other hand this stack based machines, so what is assumed that is that so there is a stack. So, whenever you do any operation, so this arithmetic logic operations, so they just pop out the operands from the stack. Do the operation and push the result back to the stack, so that way the code generation for this stack based machine is definitely different from the code generation for the accumulator based machines.

Then this CISC and RISC architecture, so they also play major role like CISC instructions so they are more complex. So, we have got a longer length instructions, complex instruction, they take different amount of time for execution. Whereas, the RISC instructions, so they are more or less similar. All the instructions are more or less similar in size and execution time.

So, as a result this CISC versus CISC and RISC instruction sets, so they will also tell like what is the amount of what is the type of code that we generate for them. So, this way and they have got many other features like the RISC machine, so they have got large number of registers compared to the CISC machines. So, naturally will have more amount of the registered usage will be more. So, optimization and everything will be at different come in RISC machine compared to the CISC machine, so that is about the hardware platform.

(Refer Slide Time: 08:11)



Operating System and System Software

- Format of file to be executed is depicted by the OS (more specifically, *loader*)
- Linking process can combine object files generated by different compilers into one executable file

26

Operating system and system software, so format of file to be executed is depicted by the operating system. So, so what happens is that we have got this operating system will be having the tool called loader. And this loader module, so it will be finding out it will have some specific format of the object file ok. So, these object file that we are having so any compiler that you have. So, it should try to generate object file in that format. So, otherwise the loader will not accept it, so the program cannot be loaded into the memory.

Then the linking process or the linker tool, it also has got a major role to play, because they will combine many object file generated by different compilers into some executable file. So, this linking or linker tool, so it will also tell the of the acceptable object file format, so this is also there. So, of course there are many different link file formats. So, normally linkers the support a number of different formats, but anyway so you it is this code generation process will be guided by the, this linking tool that we have.

(Refer Slide Time: 09:25)

Error Handling

- Show appropriate error messages – detailed enough to pinpoint the error, not too verbose to confuse
- A missing semicolon may be reported as “<line no>; expected” rather than “syntax error”
- If a variable is reported to be undefined at one place, should not be reported again and again
- Compiler designer has to imagine the probable types of mistakes, design suitable detection and recovery mechanism
- Some compilers even go to the extent of modifying source program partially, in order to correct it

Handwritten annotations on the slide include: a circle around '8i' with an arrow pointing to 'if'; a circle around 'int' with an arrow pointing to 'x' and a cross; and the code snippet 'y = 2 * z' followed by 'x = x + 1'.

The slide footer includes the Swamyam logo and a small video feed of a man in a purple shirt.

Then error handling. So, you have to show appropriate error messages. So, this is very important as I told in the last class. So, for the user to understand and to appreciate that this compiler is doing a good job. So, the user may be asking for user may be looking for more meaningful error messages rather than syntax error message like syntax error at line number 13, so that is very that is that is very cryptic. And the user will not be able to do much with that or even if at least the line number has to be mentioned. So, so if the line number is also not mentioned, then it is it may be a message simply that syntax error. So, syntax error where is the syntax error that you do not know, so that makes it difficult.

So, they must be detailed enough to pinpoint the error. So, this error message that is the flash, so it should be detailed enough. However, it should not be very verbose to confuse, like say maybe the compiler tries to give two three different options by which the error might have occurred, so that way it is it sometimes it is more confusing. So, so those things are to be avoided.

So, this is actually up to the compiler designer to decide like what should be the appropriate error message for different types of errors, and that way compiler quality will be just based on the type of error message that is given. So, missing semi colon may be reported as line number, and then followed by semi colon expected rather than syntax error. So, so the if you give this expectation part, what is the, what was the expected

symbol here. See if you mentioned that, then definitely it is easier for the user to correct the program.

Another example is like this variable may be reported undefined at one place; it should not be reported again and again. So, what I mean is that suppose, I have got a piece of program, where at this point I have got the variable y equal to x plus z . And I have forgotten to put this declaration of x somewhere here. So, this declaration should have come before the usage of x , but somehow it is missing it is not there.

Now, in some later lines also I might have used this x equal to say x plus p that way x might have been used several more times in the piece of code. Now, so it is so once this at this point we have detected, that there is an x which is undefined. So, you have flash this message that x is undeclared or undefined. Then there is a no point, repeating that message at this point as well ok.

So, so if a variable is reported to be undefined at one place, should not be reported again and again.

So, to handle the situation what the compiler can may do is that compiler may make a temporary entry into the symbol table tell the with that with an indication and with a indication that this is actually undefined this is entered by the compiler. So, this accordingly the later part. So, this when it searches the symbol table, so it will find x there, so it will not be a undefined variable, so like that so this can be done.

In compiler designer has to emerging, so this is the very important point that I wanted to emphasize. The designer has to emerging the probable types of mistakes, and design suitable detection, and recovery mechanism. And that is the expertise or experience of the compiler designer. So, what is the type of error that that any user may make while writing program in a particular language, so that has to be imagined by the compiler designer.

And some compilers even go to the extent of modifying source program partially, in order to correct it. So, this is similar to say the corrections that we that is done by many of the word processing software's while we are entering the text. So, it does some it replaces the wrong words by correct ones. So, whether we like it or not that is the different issue, but sometimes it is very helpful. So, similarly when we are writing a

piece of program for example, if I writing a C language program if at the beginning of a sentence if I write fi, then most possibly most probably I am actually I tried to write if and change it to I fi.

So, compiler can do this type of changes. So, it can change fi to if and when it finds that there is some errors some error is occurring then say fi, so fi will be taken as a variable and the variable if it is not available in the syntax in the symbol table; that means, fi is a fi is not a variable in that case may be the user has done a mistake and instead of writing fi if has written fi. So, this compiler can be can correct that fi to if. So, this type of corrections can be done by the compiler itself, so that is one possibility.

(Refer Slide Time: 14:27)

Aid in Debugging

- Debugging helps in detecting logical mistakes in a program
- User needs to control execution of machine language program from the source language level
- Compiler has to generate extra information regarding the correspondence between source and machine instructions
- Symbol table also needs to be available to the debugger
- Extra debugging information embedded into the machine code

Handwritten notes on the slide include: $y = x/z$ with x, z next to it, and $x = f(\cdot)$ with an arrow pointing to the list item about the symbol table. There is also a small diagram of a computer monitor and keyboard.

Then the other thing that we have is the help in debugging. So, debugging is suppose we have written a program. So, it is syntactically correct, but there are some logical errors in the program. So, it may so happened that in a program I am writing I have somewhere I have written like y equal to x by z. And we find that it is giving some result which is undesirable, and one possibly so maybe we want to we want to check at this point what are the values of x and z, so that whether the value of y is computed properly or not.

Or suppose we have called a function x equal to f some function, now after this function has been executed, we need to we may want to check the value of x to see whether the value computed is correct or not. Now, apparently we can do all these things by

introducing some print or write statements in my program at some intermediary places, but it is very cumbersome.

So, what the user can think about having a facility like I should be able to run my program line by line after every line the execution should suspend should get suspended, and I will be able to check the values of the variables, and then again give the continue signal. So, that this execution continues.

So, this way the user needs to control the execution of machine language program, but sitting at the source language level. So, the user does not know machine language level. So, what is the translation of each and every instruction? So, if the machine if the processor stops after every single program line execution, so that becomes difficult. But so that is of no use to the user because user will not be able to track at that level. So, user can track at the source language level, so that way we need to have some information where exactly to stop.

So, what you get is that if this is a source language program, and this is the corresponding machine language program, then after every statement that we have in source language I should. So, if this source language program translates to this much of code in the machine language then after this I should put some additional information here, so that I can suspend execution at this point. Similarly, after these are next piece of code starts here, then after that there should be some facility some code must be sitting here, so that I can ask the compiler ask the execution process to stop at that point. So, this is normally done by the help of a debugger this stopping execution in between and all, but the compiler needs to introduce these codes into the machine into the machine language program, so that is very important.

So, compiler can help in the debugging process by providing this information in terms of these that can help the debugger ok. So, compiler has to generate extra information regarding the correspondence between source and machine instructions. And symbol table also needs to be available for to the debugger, because debugger it will be the user will ask for values of different variables and debugger needs to find out the offset of those variables. And offsets are available only in the symbol table, so that way this offsets will be used by the debugger from the symbol table. So, symbol table is going to

be preserved in this case though previously we say it that in the target code symbol table does not have any role, but here it has got some role.

So, for if the program is to be in debug mode, then this symbol table is useful. So, this is particularly true when we have got initial phase of development. So, initially in the program there will be lots of bugs logical bugs. So, that way we need to debug our program mode and once we get the confidence that my program is now logically correct, so I can I can remove all this debugging information and then the I can generate a code which will be running more efficiently.

So, this there are two distinct situation in which the compiler is made to run; in one case it is generating lot of debugging information and it does not do any optimization there; in another situation it does not generate any debugging information, but does lot of optimization, so that the program runs efficiently. So, this so this debugging and optimization, so they go both of them are required in the program development phase, and the compiler should be able to handle that situation.

(Refer Slide Time: 19:19)

Optimization

- Needs to identify set of transformations that will be beneficial for most of the programs in a language
- Transformations should be safe
- Trade-off between the time spent to optimize a program vis-a-vis improvement in execution time
- Several levels of optimizations are often used
- Selecting a debugging option may disable any optimization that disturbs the correspondence between the source program and object code

Handwritten notes on the right side of the slide:

$$\begin{aligned} &R = y + z \\ &P = x \times y \\ &Q = x \times R \end{aligned}$$

The slide is part of a presentation, as indicated by the 'swayam' logo and a small video feed of a speaker in the bottom right corner.

So, my compiler should provide these facilities then optimization. So, naturally so these are the series of transformation that may be necessary that may be beneficial for most of the programs on in a language. So, the compiler designer must understand the general set of optimizations that are that are useful general set of transformation that are useful for most of the programs in the language. So, guessing this particular say it is not very easy,

then the transformation should be safe. Safe in the sense that if I do some optimization, so it should not change the meaning of the program.

For example, suppose I have got a statement like $x = y + z$, and then at different places I have got reference to this x ok. So, at this point, I have got a reference to x $p = x + \text{something}$. So, so then again at some point later I have got $q = x \text{ into } r$. So, like that now this x is a variable. So, it is kept in the memory.

So, if two in order to reduce the load on execution time maybe we do not load this x from memory again rather so this x may be copied into some CPU register as well and that at this point. So, we take the value from the CPU register, so that way it so one memory, memory access is set. So, if I have got multiple access to x in the program so and if I have a register that holds the value of x then that should be, so will save so many memory accesses.

But the point is that the register that we are putting here remains reserved for the entire sequence. So, whether that is really the case or the register value can get modified due to some program execution sequence, occurrence of interrupts etcetera, so that may make this transformation unsafe ok so, that the program may not work correctly under those situations. So, that is what it says that the transformation should be safe. So, we should do optimize, but optimization should be safe.

Then the trade-off between the time spent to optimize a program and that time improvement in the execution time, so that is that trade-off is necessary. But this so the amount of time to optimize a program and this if you want to do more optimization, it will take more time, but it will lead to more improvement in the execution time.

Then there are several levels of optimizations are used. So, you can have as I said that their, their, their different levels we can do optimization up to different extents. So, you can have different levels of optimization. And as we go deeper and deeper or higher and higher level of optimization, the compilation time increases, but the amount of optimization that we get the efficiency of the code that also improves. If we are doing some debugging if we selected debugging option, so that may disable any optimization that disturbs the correspondence between the source program and object code, because optimization may change the sequence of instruction execution. So, naturally if you have

your selecting debugging mode or debugging option, then the all the optimizations are to be turned off, so that is there. So, these are the various issues that we have.

(Refer Slide Time: 22:57)

The slide is titled "Runtime Environment" and features a yellow background with a blue border. It contains four bullet points: "Deals with creating space for parameters and local variables", "For languages like FORTRAN, it is static – fixed memory locations created for them", "Not suitable for languages supporting recursion", and "To support recursion, stack frames are used to hold variables and parameters". Handwritten notes in the bottom right corner include the expression $x = f(a_1, a_2)$, the text "function f(r1, r2)", and "int b1, b2". The slide also includes a logo for "swayam" and a small gear icon in the bottom left corner.

- Deals with creating space for parameters and local variables
- For languages like FORTRAN, it is static – fixed memory locations created for them
- Not suitable for languages supporting recursion
- To support recursion, stack frames are used to hold variables and parameters

$x = f(a_1, a_2)$
function f(r1, r2)
int b1, b2

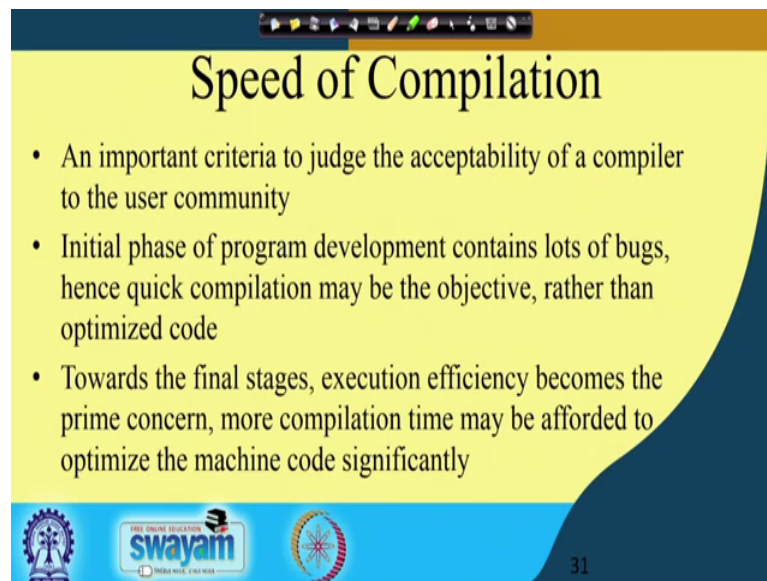
Then the runtime environment. So, deals with creating space for parameters and local variables. So, like say if I have got a piece of program, and at some point I am calling a function f with say write a x equal to f with some parameters say a_1, a_2 etcetera. Then in the function f in the function f I may have some more local variables. So, this is r_1 and r_2 , these are the two arguments.

And maybe I have got some local variables b_1, b_2 etcetera. Now, where are these $a_1, a_2, r_1, r_2, b_1, b_2$ stored that is the decision that we take in the runtime environment management, so that will be the reserving space for parameters and local variables. So, for languages like FORTRAN, so this is done statically. So, we have got fixed in memory address memory locations were created for them from where it will take up all these values ok.

So, this is this makes the compiler design process simple, but at the same time it creates difficulty if you are going to support recursion, because recursion the same program may be called again. So, the previous locations which we are reserved for $r_1, r_2, b_1, b_2, a_1, a_2$ etcetera they will get over written by the new call and the new creation of the next instance of the call to the function f .

So, this makes it difficult for supporting recursion. For recursion supporting, so we have to have some sort of stack frames, so which are more complex in nature compared to this is static runtime environment management. So, we can have to go for that. So, when you go to the runtime environment management techniques, you will see that they are very efficient technique, that have been developed for handling this recursion.

(Refer Slide Time: 25:05)



Speed of Compilation

- An important criteria to judge the acceptability of a compiler to the user community
- Initial phase of program development contains lots of bugs, hence quick compilation may be the objective, rather than optimized code
- Towards the final stages, execution efficiency becomes the prime concern, more compilation time may be afforded to optimize the machine code significantly

Logos: AICTE, swayam, and a circular logo with a gear and a person.

31

So, next we have got a speed of compilation. So, this is another important criteria to judge the acceptability of a compiler to the user community that is how fast is the compiler working. So, it is a flash it gives the output is the give the compiled output or it takes lot of time. So, so that is that way it is difficult ok. So, initial phase of program development, it contains lots of bugs, hence quick compilation may be the objective rather than optimized code. So, they are also most of mostly we have got syntax error, then this logical bugs and all. So, there is no point doing optimization there.

So, at that that time I should have an option by which I can tell the compiler see I do not want any optimization I want the compiled code fast ok, so that should be generated quite fast or if there is any syntax error that should be reported fast. So, that part the initial phase of program development. So, we are we are will be looking for the compiler to work with the high speed of compilation maybe we forgo the optimization part.

But when you go to the final towards the final stages execution efficiency becomes the prime concerned, because there we are ready to spend more time, but we do not want the

compile that to the code generated to be inefficient, so that the code has to be very efficient. So, we want to spend more time on the compilation process, but we want that the output or the outcome of the compilation, it should be very efficient, the code generated should be very efficient. So, this is towards the final stage of compilation process we want optimized code.

So, for the same compiler sometimes we want that it should be running very fast, sometimes will be we will want that it should be not it need not be that much fast, but it should generate very efficient code, so that is these are the issues that we have with a compiler. So, the compiler designer should look into these angles. So, not only the code generation process, but this entire other issues that to be looked into and that makes the compiler design process complex.

So, rather than a say if you do not bother about all this concerns, then it is pretty easy. So, we have got automated tools by which we can generate the code for some for some processor, but as soon as you bring all this concerns into your purview, so this entire code generation process becomes difficult.

So, in our course, so we will try to look into these aspects again and again in different chapters and that way it will go. So, at the end we should be able to we should be able to design compilers within reasonable amount of time and taking care of all these issues.