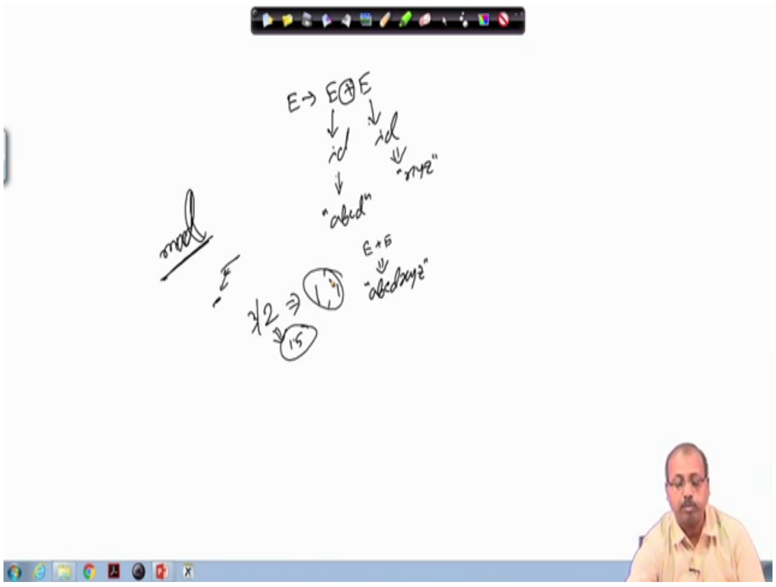
**Compiler Design**
**Prof. Santanu Chattopadhyay**
**Department of E & EC Engineering**
**Indian Institute of Technology, Kharagpur**

# Lecture - 39
# Type Checking

The next part of our course, it will be talking about Type Checking. So, type checking is very important because what happens is that in most of the programming languages, they use variables and expressions they have got some types. And most of the time this with this depending on that type so certain word lengths are allocated to individual types. For example, in many compute many computers you can find that integer is given 2 bytes or floating point numbers are given 6 bytes. So, like that. So, if the type is not correct, then this computation becomes difficult. So, that is one thing.

Second important point is that see whenever I have got an identifier. So, we can the grammar rule says that, if I have got say E producing id. So, E producing id plus id something like that then

(Refer Slide Time: 01:10)

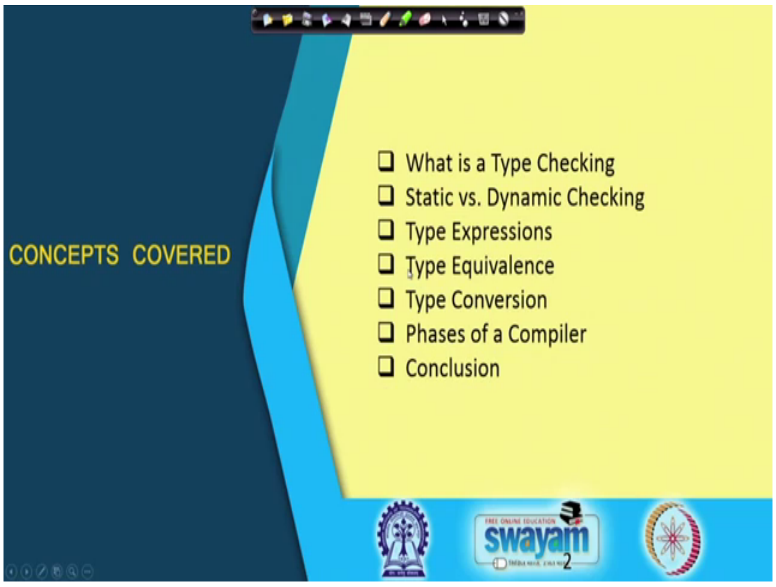


So, grammar rule has told me that we have got this type of rule. So, E producing E plus E and we have seen that this E can give me id this E can finally, give me id. Now, but this plus operation; so it is not defined on different types of identifiers. For example, if it is integer integers, then this id plus id has got a meaning. But if it is say for example, say character variable. So, what do you mean you what do you mean by this plus. Similarly sometimes it may be meaningful, but in a different context. For example, if this id so, they are corresponding to some string. So, maybe the first this is one string. So, abcd and this is another string say xyz.

And in that case this E plus E. So, this E plus E may mean that this is the concatenated string abcd xyz whereas, the it really a totally different meaning when you have got this either expression or these ids as integers. So, this way we need to check like what which operation that will be that is we are that the programmer is willing to say whether it is a string operation or whether it is a integer operate and integer operation. So, like that is one thing and many things are not allowed also on different types for example, if we say one many of the computer programming languages they will support the modulus operator. So, modulus operator is say it will do an integer division and as a result. So, this will give me the reminder part.

So, this after division the result and the reminder so, they are often true when we have got the integer arithmetic. So, when you are dividing 3 by 2. So, result may be 1 and with a remainder of 1. So, if it is an integer division, but if it is a if these are taken as real numbers, then the answer should be 1.5. So, that way it is whether I am going to follow a real d real number division or an integer division. So, that is determined by the type of the operands. So, it is very important that we should know that types not only for the for determining the size of individual of a identifiers, but also to know whether certain types of operations are allowed in that on that particular structure or not. So, that is that these are the two things for which we will see this compiler designers. So, they will be using this type checking.

(Refer Slide Time: 04:04)



CONCEPTS COVERED

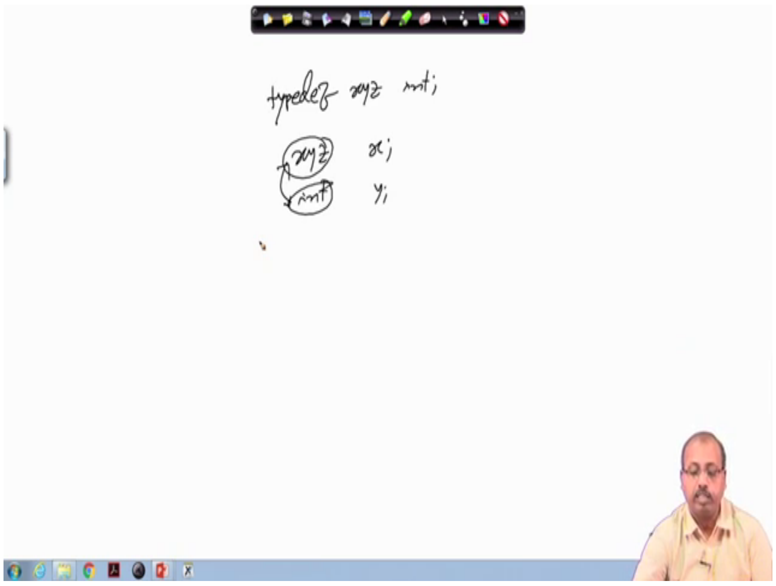
- What is a Type Checking
- Static vs. Dynamic Checking
- Type Expressions
- Type Equivalence
- Type Conversion
- Phases of a Compiler
- Conclusion

So, topics that we are going to cover are like this. So, we will see what is type checking that is the first thing, then this type checking may be static or may be dynamic. So, static means so, it is done at the compile time itself and dynamic means while the program is in execution, we do the type checking. So, both have got their implication. So, step most of the programming languages they will go for static type checking, but many a times we will see that some programming languages they have allowed dynamic type checking. So dynamic type checking so, that is the help full because the same variable x defined in two different contexts. So, they may have different bearing. So, in one case if x may be considered as an integer, in another case x may be considered as string variable. So, like that it can happen. So, we can have this static versus dynamic checking.

Then there are for type checking so there is something called type expression. So, if you are given a complex structure like, if you look into the C programming language then you see they we have got this structure and union type of declarations where a number of data items are combined together to make the overall structure. So, what is the type of the overall structure or when I have got an array of fundamental types like the array of integers and array of real numbers like that. So, this array of the certain types of that also creates a new type so, that is an array type. So, that way so, that is also some new types. So, how to formalize this whole form whole type construction phase; so, that will be the type expression.

And we need to check type equivalency very often we need to answer this question whether this x and y their types are equivalent or not, they might not have been defined to be of same basic type, but it may so happen that the user might have redefined as the integer as another type like we can always say in C language you know that there is a typedef type of thing.

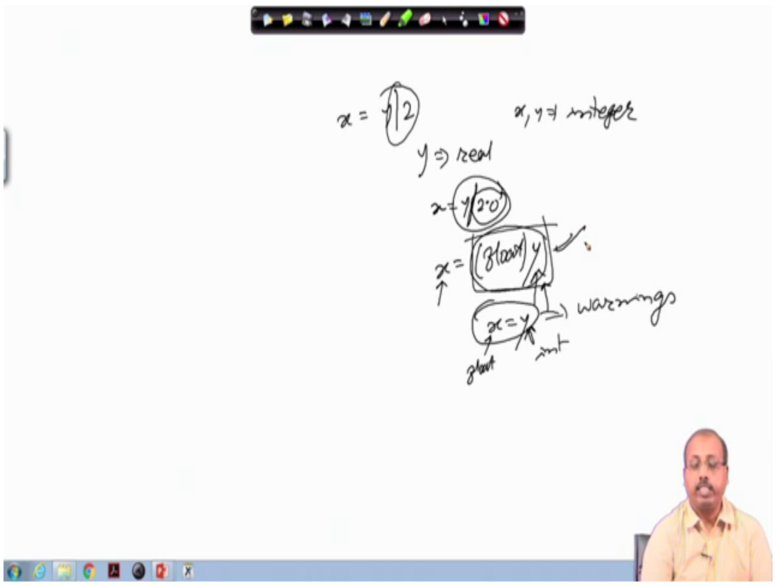
(Refer Slide Time: 06:15)



```
typedef xyz int;
xyz x;
int y;
```

So, we can write like this typedef say if say xyz integer. So, later on if I have a variable x y z so, a variable x of type xyz and another variable y of type integer. So, are they equivalent or not? So, this is a very difficult question to answer like if you just look at this xyz so and this int so, they are of different types, but if we go further. So, we can see that they are derived from the same basic types. So, they are going to be equivalent in that case. So, whether we go to that level of equivalency check or not that is a different question. So, that is often dictated by the programming language, but if it has to be supported then, it becomes difficult because we have to do it more rigorously. So, this type equivalency we have to do. Then many times you will need type conversion.

(Refer Slide Time: 07:22)



So, type conversion is like this that suppose, I have got an operation in suppose I am writing a program in C language and where I am writing say x equal to y divided by 2 and here if this x and y they happen to be integer, then this operation is an integer division operation. So, this is taken as an integer division whereas if this y happens to be a real number then this is taken as a, this division is taken as a real division real number division.

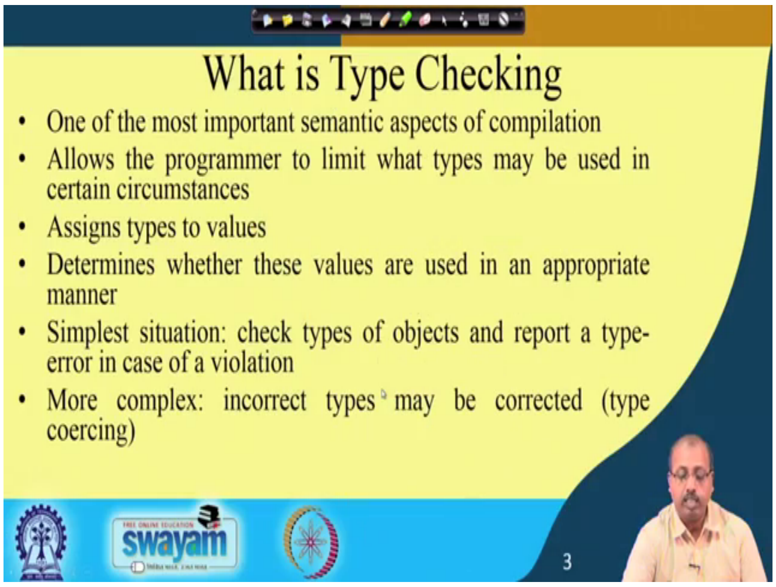
And many a times what will happen is that you even if y is of type integer so, you can write like x equal to y divided by 2.0 and since this 2.0 is of real is a is of type real then, this y divided by 2.0 so this whole division is done in real number division not an integer division. So, this way so, locally we may need to change the type. So, convert the type of an expression from one type to another type. And we have also have got casting of types. So, for example, I can write x equal to float of y in say C language where this x may be a floating point number, but y may be an integer. So, when I say that float y so, this y is converted locally into a floating point number and then it is assigned to x.

So, naturally so, the type of this right hand side expression is becoming floating point. So, if I write simply x equal to y then, if x is the x is a floating point number and y is an integer. So, this is clearly a type mismatch because that is outcome of this is often dependent on the compiler and the underlying system like how this integer and floating point numbers are represented in the underlying machine, so underlying computers, so based on that this result will be determined and particular in particular the sizes of x and y will not be same. So, which part of x will get the value of y? So, that becomes a concern. So, if x is say 6 byte and y is 2 bytes. So, where does these 2 bytes of y go? So, that is that may not be very clearly defined ok.

But as soon as you say that this is as soon as you write it like this x equal to float of y. So, then you are very clear in your mind that I first want to convert y into a floating point number and then use it for this assignment. So, many cases if you write like this. So, compiler designer, they should give a warning. So, you might have seen this type of warnings that there is a type mismatch in this assignment whereas, if you do it like this then the compiler will know that you are really trying to do a local conversion of type. So, as a result no warning is generated. So, this comes as a responsibility of the compiler designer to check that type of individual identifiers and accordingly the type of expressions.

So, we will see how this can be done. Next so phases of a compiler. So, this will be where we will see this in the particular this type conversion coming into picture and then we will draw the conclusion.

(Refer Slide Time: 10:47)



## What is Type Checking

- One of the most important semantic aspects of compilation
- Allows the programmer to limit what types may be used in certain circumstances
- Assigns types to values
- Determines whether these values are used in an appropriate manner
- Simplest situation: check types of objects and report a type-error in case of a violation
- More complex: incorrect types may be corrected (type coercing)

3

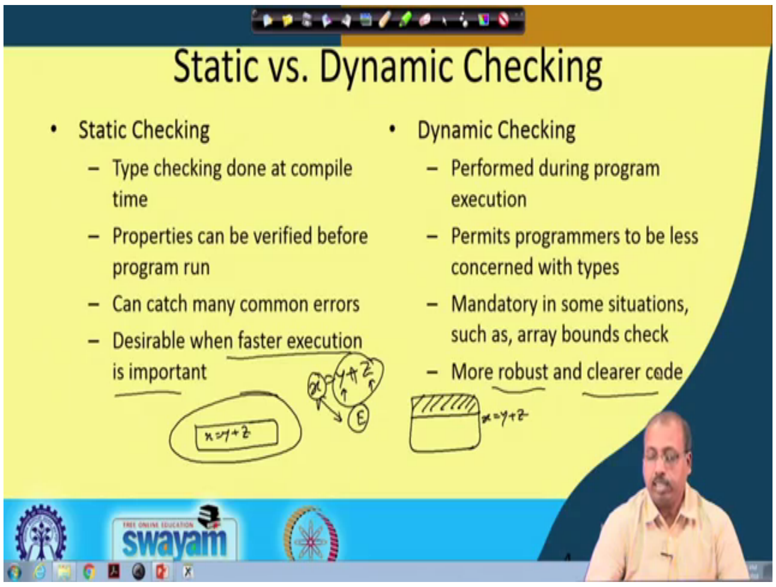
So, try to answer this fundamental question like what is type checking. It is one of the most important semantic aspects of compilation. So, please note the term that this is the semantic aspect, it is not the syntactical aspect. So, you cannot use the standard automata theory to answer whether a program is correct from there from its type point of view because of the reason that automata theory can say that this is an identifier the token types and all the tokens and all what are the different tokens that are present in the program accordingly whether the combination of those tokens make follow some grammar rule or not. So, those things can be done by your lexical analyzer and parser, but it cannot determine type of expression automatically unlike the parsing which can determine the syntactical errors very automatically. So, there is no automated way to determine this type types and all. So, this that is why it is a semantic aspect it is not a syntactical aspect.

And it allows the programmer to limit what types may be used in certain circumstances. As I was telling, that certain operators so they are applicable on certain operand types only. So, if you look into the programming language manual so, it will tell you like for different operators; so, what are the expected operand types. And then, so accordingly the programmer has to follow that, but who will check this thing? So, this has to be checked by the compiler not as part of this syntax analysis, but as part of the semantic analysis. So, this type checking process it will assign types to values. So, as you are writing expressions so, it will be assigning some types to those expressions and determines whether these values are used in an appropriate manner. So, if we represent types by values then whether this values are going to be used like say when I am having this an addition operation, I expect that the operands are to be of type say integer then whether we have got the integer operand or not for both the operands of this addition operation. So, that has to be checked. So, that is the in appropriate manner whether the operands have been used or not.

The simplest situation is like this, the check types of objects and report a type error in case of a violation. So, as I was telling so you check the types of individual identifiers as they are occurring and if you are finding some mistake some problem as I was telling that if I have if you write like x equal to y and the types of x and y are not same then this is a type mismatch. So, if we have got a very strongly typed language. So, that will not allow you to have this type mismatches. So, that is an error. So, if the compiler should flash an error that this is there is a type mismatch at such and such line.

However, the more complex situation comes when you have to correct the type like see you find that there is a type mismatch, but you locally change the type of y to so, that it matches with the type of x. So, if you do this thing then that is known as this type coercing. So, you locally change the type of some of the some part of the expression and so, that this operands so, they are having their correct types. So, this is more complex because the compiler designer need to understand like what is the expected type and all and how to do the conversion etcetera. So, do we doing it correctly is may also be a challenge like how to how to get a good type checking type coercion done.

(Refer Slide Time: 14:48)



## Static vs. Dynamic Checking

- Static Checking
  - Type checking done at compile time
  - Properties can be verified before program run
  - Can catch many common errors
  - Desirable when faster execution is important
- Dynamic Checking
  - Performed during program execution
  - Permits programmers to be less concerned with types
  - Mandatory in some situations, such as, array bounds check
  - More robust and clearer code

So, once that is. So, next we will be looking into this static versus dynamic checking. So, type checking is important, but when do you do the type checking? So, one type of approach is to do static checking. So, type checking is done at compile time itself. So, at compile time itself all the variables and identifiers. So, they will have their types known and we can check their type at the compile time only. And in case of dynamic checking, so this is performed during program execution. So, as you can understand so, this is more complex the dynamic type checking.

So, this whenever you are going for program execution, so at that time so you need to do the checking. So, the compiler designers job is more critical in case of dynamic checking because in case of static checking what the compiler designer will do is that if this is the description of some this is some program. So, it will check line by line and say at this point it has got say x equal to y. So, it will immediately know what is the type of x, what is the type of y accordingly it will take a decision; whether it will say that it is or whether it will say that there is a type mismatch or it will do a coercion or not. So, those are secondary issues, but it is the compiler designer can do that thing at this point.

But the same thing, when you are doing in a dynamic type checking the situation is more difficult because here for x equal to y so, the this compiler for under static type checking. So, compiler designer might have generated the target code where say this part of the code was doing that assignment x equal to y. And since it is static type checking so, the compiler designer knows that if the code has been generated properly. So, this x and y they are of a similar type. So, there is no type problem. But in case of dynamic type checking, what is happening is that so here, at the when you are generating the code. So, suppose this is the code in that code so apart from that assignment of x to y. So, you have to also have the code where the type checking will be done. So, you should have some code here which will be doing the type checking.

So, that way it becomes costly at the in the dynamic checking philosophy. So, this additional code has to be there in my along with this assignment. So, that it will; here it will do the check for x and y and as defined in the programming language so, it may the program may generate an error if these types are not correct or the program may do something that the execution should do something so that it this it is taken care of. So, that particular check is done by this piece of code. So, if it finds so, after analyzing the types dynamically at the runtime so, if it finds that the x and y are of different types. So, as it is defined in the programming language it will take a it will take an action.

So, if the programming language says that this is an error so, the execution will abort. And if the if the programming language says no it is correct it can you can continue so, they may be it will locally do some conversion. So, those things will be done in this piece of code. So, you see their dynamic checking. So, it will necessitate to have additional code in the target code that is produced.

Now, then why should we go for this? So, dynamic checking; so, if it is so complex then why you why are you going for this? So, this is important because it is it may be the case that statically if we bound it then a particular function if it is called from different contexts the its variables may have different meanings. So, that will not be possible with static checking because then because the actual call is happening dynamically ok. So, that has to be taken care of by the dynamic checking only. So, this so, that is why the dynamic checking is done in several programming languages, but of course, it makes it a bit difficult to generate code and a life of the compiler designer becomes a bit tough. Anyway so, this static checking: so, this is type checking will be done at compile time and in case of dynamic checking this checking will be performed during program execution.

So, in static checking, properties can be verified before program run. So, whatever be the type checking property so, all those checks can be incorporated. As I was telling that if some operation is being carried out whether the operands are of correct type or not then so, all those checks can be done before the program is run. However, in case of dynamic checking, so, this will not do these things. So, it will permit programmers to be less concerned with types. So, programmers can be made free like if you compare between say the languages like say C and Pascal. So, if Pascal is a language where it is strongly typed. So, all variables they are having their types and all and the types cannot be changed and all, but in case of there are certain operations that that can be done on a certain types of variables only. But in cases c there are much more flexibility. So, though it is also first typed language, but it is not as strongly typed as Pascal. So, but so, that way the programmers are a bit free to have to be less concerned about the types.

So, whether it is good or bad that is a question like it may be good because programmer has got more flexibility. So, like whether I have defined all the variables there types or not so, that may not be a concern, but at the same time it may be a problem because the program may generate error and basically the unforeseen type of error. So, which the programmer might not have thought about like; it may give some error due to this type problem and it is very difficult to catch this type of errors in program run. So, static checking it can catch many common errors. So, what are the common errors like we have got some variable types, they are type there is a type mismatch and all so that way though they are very common error. So, this compiler can very easily catch those thing.

Now so, this that way the static checking can do this thing. It can do this check and all. Then dynamic checking, so this is maybe because this type of common errors like whether this x equal to y the x and y is type are matching or not. So, that is not checked that is checking it dynamically is difficult, but many times what happens is that we need to have this dynamic checking also. So, one such example is the array bounds check. So, if we look into this for example, suppose I am writing in a in a program a line like a i equal to some expression.

Now, I have defined a to be an integer array a 1000. So, naturally if the value of the i and the I mean that this a array a is the valid indices at 0 to nine 999. So, these are the valid indices for a. Now this if this value of i is less than 0 or it is greater than 999, then what fine? Now so, in that case what is going to happen? So, statically you cannot check this because until and unless the program has run, you do not know what is the value of i. So, there is no question of statically checking whether this index is within the limit or not; that is not possible. So, dynamically we can do that because when the program is executing. So, this is my program. So, when the program is executing at this point I know what is the value of i. So, if I know the bound of this array a that its minimum index is 0 and maximum is 999. So, I can have a check like this.

So, before doing this assignment so, if the if I for this there doing this assignment if this is the code which will be doing the assignment on top of that I can add another piece of code which will be checking that if i is less than 0 or i is greater than 999 then, it is an error. So, I can call some system error function and that will flag some error message and maybe the program will get aborted. So, like that. So, that way it is done. So, this array bounds check is a very important thing because it will it is really helpful because the programmer while writing the program might not have been very careful to ensure that this array index i does not go beyond its limit.

Now, if we have this type of check in the code that is generated so, the some programming language will tell you that you should have this array bounds check, some programming language will tell you that no there is no array bound check. For example, if you look into language like Pascal, you will see that this array bounds check is there whereas, if you look into language like C. So, C will not ask for this array bound check. Why it is not asking for array bound check? Of course, there is a reason because this C language so, it will allow to access a i by pointers also like you can always say like p equal to star a p equal to says suppose I have got an integer array a 1000 and there is an there is an integer pointer p. So, I can always write like this that p equal to a and then this star p plus i equal to something.

Now, this i when even if you know the value of i so, what do you do by that? So, star p plus I means if this is the location pointed to by p. So, it will you go I locations ahead and try to modify this. So, whether this modification you are doing as part of a array or independent of a array so, that is not known. So, the C language tells that I will not do any such dynamic checking because they just to support this type of pointer arithmetic so they have removed this. So, whether this. So, you see that it if sometimes becomes risky because if this value of i is say, 1010 then what will happen? If this is the array a and it ranges from your 0 to 999. Now when you are saying like so you have set the p pointer here and then you are saying this p plus 1010 and then put a star over that so; that means, it will be modifying some location here whatever it may mean whatever that address may mean. So, it may be within the address space of the program, it may be outside the address space of the program. So, it will try to modify this particular memory location.

Whereas this is not possible in languages a strongly typed language like Pascal where you do not know do you do not have this type of pointer based access. So, if you even if you write say a 1 0 1 0. So, in case of C language this will be correct whereas, if you are trying language like Pascal so this will be this will not be allowed ok. So, it depends on the programming language. So, whether it is this array bounds check will be there or not so, it is dependent on the programming language.

Now, so this static checking it is desirable when faster execution is important because you see if you are doing dynamic checking as I have said that for every assignment and this operation so, you need to check the types of operands and that will make the program inherently slow. Like if I have to do say x equal to y plus z, then you need to check first whether this y and z they are of some type which supports this addition operation. So, that is the first thing to check and after you have done this addition so, you have got the expression E. Now this whether this x and E are of similar type or not so, that you can do this assignment. So, that also has to be checked and in the code that I have for this x equal to y plus z. So, if this part of the code is doing this x equal to y plus z. So, a part of it is dedicated for doing these checks.

That whereas, in this for the static checking I will not have that. So, I will only have the code which will be doing this x equal to y plus z. So, that part knowing the fully well that the compiler has statically checked the types of x y and z and found that they are ok. So, this way you see that this static checking. So, this it may be it is desirable because the faster it can lead to faster execution whereas, dynamic checking.

So, this is the execution will be slow, but it will be more robust and the code is clear like in a C language as I was telling that array bounds check we cannot have because it does not have this dynamic checking. So, since dynamic checking is not there so, the code becomes a bit clumsy at some point sometimes; it may behave in a different fashion which is unexpected by the programmer whereas, if the dynamic checking is there then all those problems can be resolved. So, you can get a more robust and clearer code with dynamic checking.