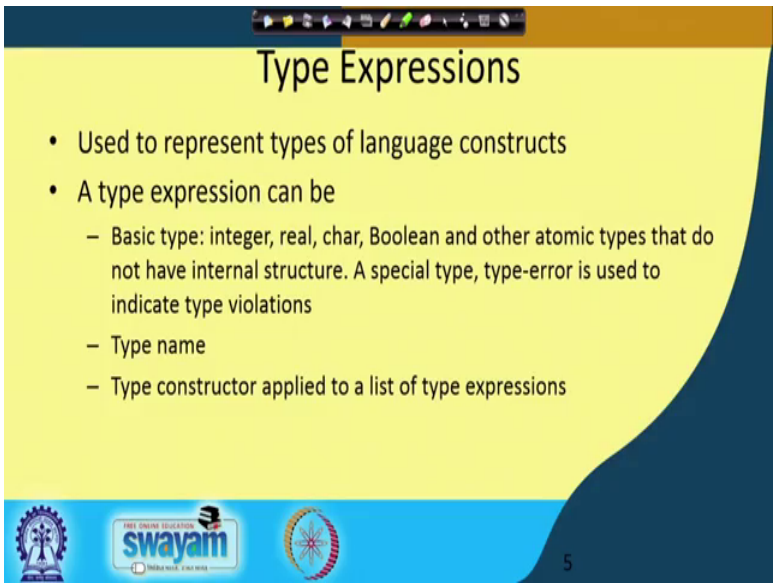
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 41
Type Checking (Contd.)

Type expressions: So, for all programming language constructs and the programming language programming elements so, will have some types. So, we can understand that when we have got some variables so, they have got a type.

(Refer Slide Time: 00:28)



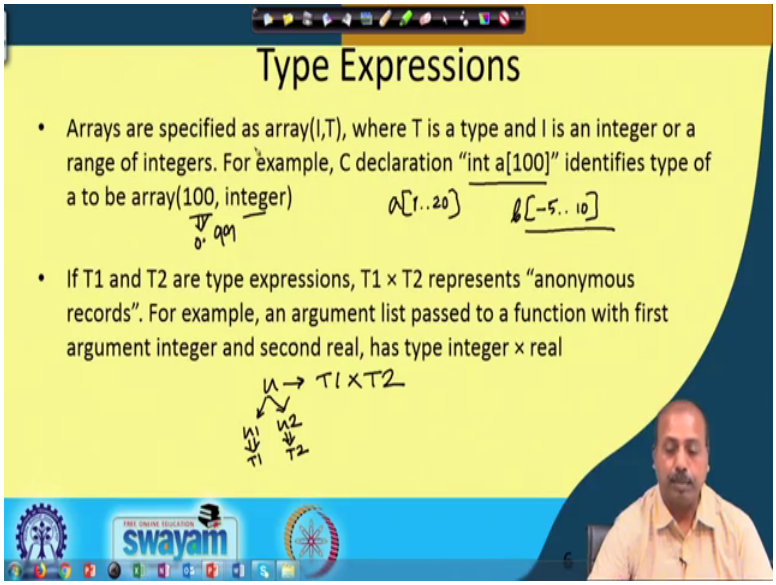
When we are talking about some say expression, the expressions have got some type, but what can what can happen is that these constructs programming language constructs so, they will also have some type. For example, some statement also can have a type a typical example maybe the statement may be taken as correct or the syntactically correct or syntactically incorrect or maybe syntactically it is correct, but in the expression in the when you go to this components of it there is some type error. So, for the whole pas construct so, we can say that there is a type error. So, that when we are pointing out some type mismatch so, we can tell that at such and such statement, so, there is a type error.

So, they are used for type expressions they are used to represent types of language constructs. A type expression can be basic type; basic type like integer, real, character, Boolean and other atomic types. Atomic type means they cannot be divided further. So,

like an integer type so, it you cannot divide it into further, until and unless now say if we say they say that there is a bit type and based on that bit the integer is framed based on the bit. So, if that is the case then that so, in that case integer also can become a derived type. Particularly, if you are looking into the hardware description languages, so, there we have got this bits as the basic type and from there we can construct integers of different bits.

So, similarly this real character Boolean so, they are all atomic types and they do not have further internal structure. So, they are the atomic type. And, there is a special type called type error so, for indicating type violation. So, whenever there is a type violation so, associated with the color correspond with the construct. So, we will put that it is type is of type error so that will can point out the type errors. Then, we can have a type name that is we can define a new type and give it a name and we can have some type constructor. So, now we will see that like array is a constructor, then your record is a constructor, so, how can we construct types from the basic types. So, that will be the constructed thing.

(Refer Slide Time: 02:46)



So, we look into some examples of this constructed types; first of all the arrays. So, array is a constructed type, like say array I, T; where T is a type. It may be a basic type or may be a constructed type or derived type and I is an integer or a range of integers. Like say in C declaration we have got say this int a 100. So, the in this int a 100 so, it identifies

the an array a to be of size has to be of type 100 integer. So, in as I said that enough it can be a range of integers also for this is particularly true in some other programming languages like C language you will not find it particularly hardware description languages.

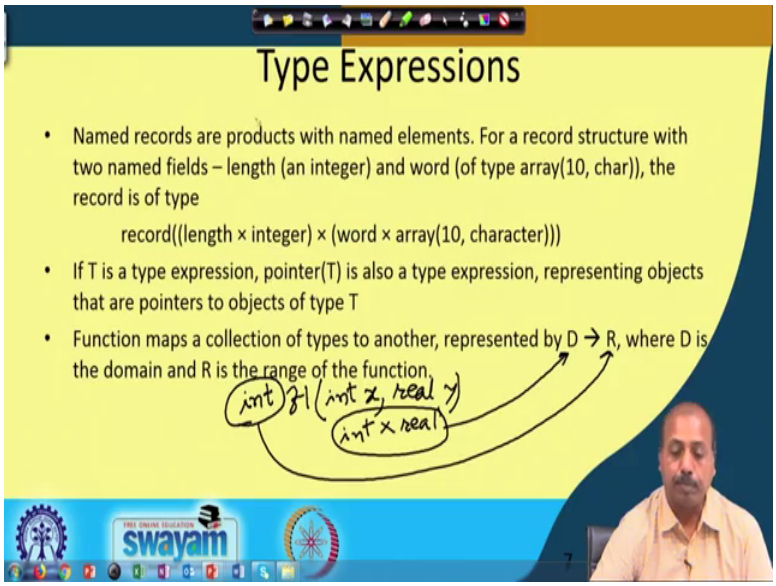
So, we have got this type of constructs like you can define an array a and you can tell the range. So, one was say 1 dot 20 so, it is saying that the lines the indices will run from 1 to 20. So, that way it is not uncommon it is not impossible to have say minus 5 dot say 10. So, this is also an array. So, we have got that into the indices will run from minus 5 to 10. So, these are all these are all possible like this range that is why it is said to be a range. So, in this case the range is range is obvious. So, it is 0 to 99, ok. It is 0 to 99, but in some cases so, it is not so obvious. So, like here it is you have to be more explicit like minus 5 to 10 so, whereas, here the range is implicit in nature. So, that is one type of constructor that is collection of basic collection of some already defined type so, elements of that type.

Then, we can have some records like say T 1 and T 2, if they are 2 type expression then T 1 cross T 2 represents anonymous records. So, if you have got any variable or symbol of that particular type T 1 dot T 2. So, like say u is a variable whose type is say T 1 cross T 2, that will mean that I will in you there will be 2 components; one is u 1 and another is u 2, ok. So, there will be u 1 and u 2 where this u 1 will be of type T 1 and u 2 will be of type T 2. So, this is of type T 1 and u 2 is of type T 2.

So, that way we can have some record type. So, is there so, this is very close like if I have got say one argument as integer and another argument as real when you are calling a function, when we are passing parameters then this argument list passed to a function with first argument integer and second real it will have the type integer cross real. So, that way we can have this type expression. So, this is unnamed because or anonymous record because it does not have a name. So, sometimes we associate some name also with this type of type this.

So, in this case so, if we have a name lab for example, the structure construct in C language so, where you have got the structures they have got some name. So, that name will get associated with the type.

(Refer Slide Time: 06:30)



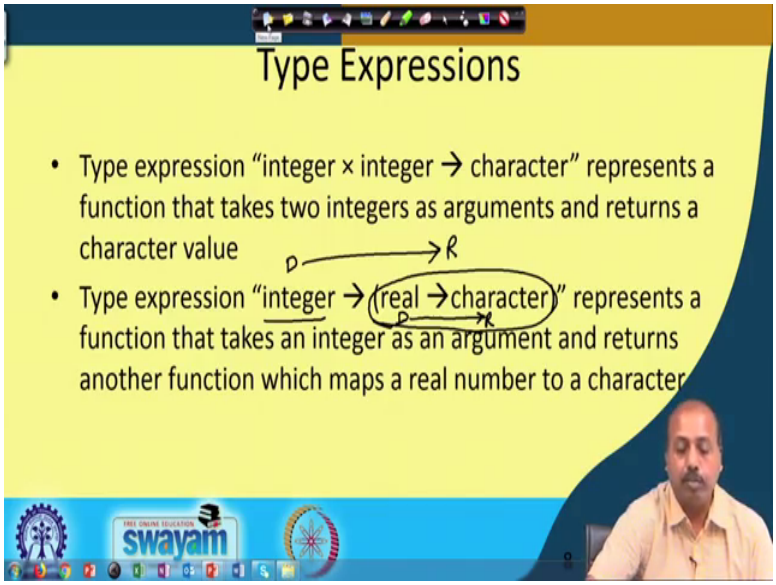
So, so, apart from that so, we have got say at named records. So, that our previously we have seen unnamed records so, we can have named records. So, these are these are products of products with named elements. So, for a record structure with 2 name fields, length and word; while length is an integer and word is an array of 10 characters. Then the record type is it is a it is type is record and it is length cross integer, so, that gives the names of the individual fields of the record and then they are corresponding types are mentioned the word and array10 character. So, that we the length is of type word and integer is an array of 10 characters.

So, it has got so, these are go so, this length is length is of type integer and what is an array of character 10. So, here this length has got that the that the first field has got the name length, second field has the got a name word and also their corresponding types are mentioned. So, this is one type of named record that can for which you can have type expression. Then, if T is a type expression, then pointer T so, is also a type expression. So, representing objects that are pointers to the objects of type T. So, this is quite common like in almost all the programming languages that supports pointers will have some way to specify this. So, here also as a compiler designer so, we have here to support this pointer type and this point at T is a type expression that will represent objects of pointers to objects of type T.

And, in case of function it maps a collection of types to one collection of types to another. So, it is a D to R where D is the domain and R is the range of the function. So, we can have something like this like if I have got a function say integer say integer f 1 and it has got arguments like say integer x and real y integer x and real y. So, here the domain that we have is basically this arguments they are belonging to this cross type integer and real. So, this is integer cross real and then that is that is the D actually. So, the integer cross real. So, this represents the D and the result is of type integer. So, this int, so, this is that R. So, R is the range of the function.

So, domain of the function means the sets from which the function can get inputs and range of the function means the set to which the function can produce output. So, we have we can have this type of function mapping, ok. So, that also so, this function f 1 is of type this will map collection of types to another and represented by D to R.

(Refer Slide Time: 10:02)



So, next we will see how this say how this integer cross integer to character. So, this is a function this is an example of a function that takes 2 integers as arguments and returns a character value. So, as I was telling in the previous example that f 1 so, it was integer cross integer cross real to integer. So, here the integer cross integer to character. So, it takes 2 integers as arguments and returns a character value.

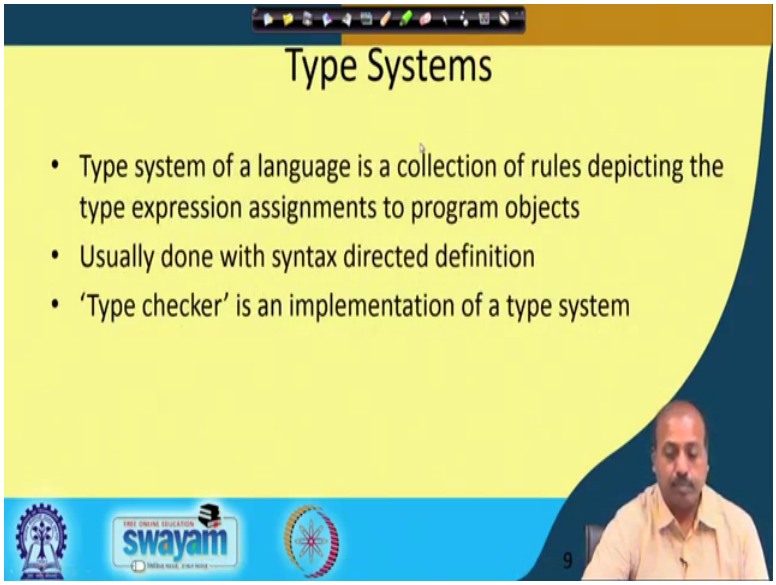
Then we can have type expression integer to real to character. So, this is basically some sort of function pointer. So, that returns a function. So, many programming languages

support this type of constructs like you can a function it can return not only some basic type basic value or these derived types, but it can also return another function as the return value. So, that function will be called when that fact that particular function will be called based on that.

So, this integer to real to characters; so, this so, this function takes integer as an integer value as an argument it takes an integer value as an argument and as a return type it returns this real to character type. So, as this thing as you can understand that if this is that domain so, it takes values from D and this whole thing is the range so, it maps this D to R. But, what is this R? This R is again real to character. So, this is again a function. So, this is also for this function the domain is real and range is character. So, it is mapping from D to R. So, that way you can say that this return type is also a function. So, from domain D integer, so, it is returning a function which can take a real argument and return a character value as the answer.

So, this way we can have these function pointers also associate with that certain types.

(Refer Slide Time: 12:10)



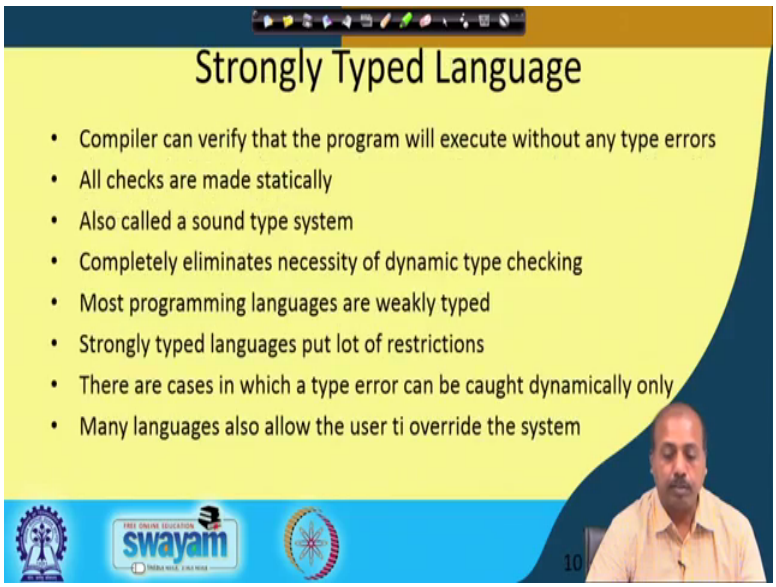
So, type system: so, this is it is a collection of rules for any programming language manual if you look into it will tell you like what are the rules of that particular language type rules of that particular language. So, this type system of a language is the collection of rules that depicts a type expression assignment to program objects like what are the

how are these objects are assigned for what will be the type rules. So, they will be collected in the type system.

And, it is usually done with a syntax directed definition. So, it is done with syntax directed definition means that you can have this type check and all and they will be implemented by means of this syntax directed transmission scheme because syntax directed translation scheme, so, it will the parse tree that is generated by the parser it will have these individual symbols. So, you can get for the individual symbols they are basic types and those types are being converted into some derived type again by some grammar rules so that while those grammar rules are applied so, you can check whether the corresponding type is constructed properly or not. And, if it is not then it will be a type error otherwise we can derive the type of the newly constructed portions.

And, type checker this is an implementation of a type system. So, we have got type system that is specified by the language and type checker is implemented by the compiler designer. So, type checker it will be using the syntax directed translation mechanism and then it will be accordingly it will be producing some actions while doing the reductions in the programming language in the in the grammar rules while doing in the parser while it is doing the reductions. So, it can check for those rules the language.

(Refer Slide Time: 14:05)



Now, if you look into the languages they can broadly be divided into 2 classes. One class is called strongly typed language where all the variables that you have all the

functions all the identifiers that you have in the in the program so, their types are well defined. So, either they are they are explicitly defined or they are implicitly defined. So, why do I say so, like in most of the languages there like say C, so, there for every variable you have to have a corresponding definition whereas, certain programming language like for example, Fortran where the first character if it is a vowel in that case it will mean that this is that this is going to be an integer. So, that way there are differences, but whatever it is there are some implicit rules and there may be some explicit definitions.

So, this compiler can verify that the program will execute without any type errors. So, while doing the parsing the entire program is known. So, it is looking into the entire program and if the compiler does not give any type error then it is guaranteed that during execution of the program there will not be any type error. So, this checks are made statically. So, they are static checks and they are also called a sound type system because this type system the type checker that we have that is very rigorous in nature. So, they are called a sound type system.

Completely eliminates the necessity of dynamic type checking because there is no nothing like no scope is left for this dynamic checking. So, it will eliminate the necessity of this dynamic type checking and most programming languages are weakly typed. So, why it is so because many times it is not possible to have everything done statically. For example, this array bounds check ok, then so, we cannot have those things that they are there need to be done statically.

So, most programming languages they are weakly type and strong because this strongly typed languages they will put a lot of restrictions on the programmers. Like whenever you are defining a function, so, you should have its type specified. Even for the language say C, so, if you define a function and do not if the function definition comes later so, if you have a program where at this point I have defined the function and say f 1; so, if I have defined the function f 1 at this point and then so, here we have got the function if 1 and somewhere earlier I am writing that code for you know I am calling the function f 1 say x equal to f 1 something like this.

Then at this point of time the compiler has not seen the type of f 1. So, it becomes difficult because what is the type of f 1. So, whether this statement is correct from the

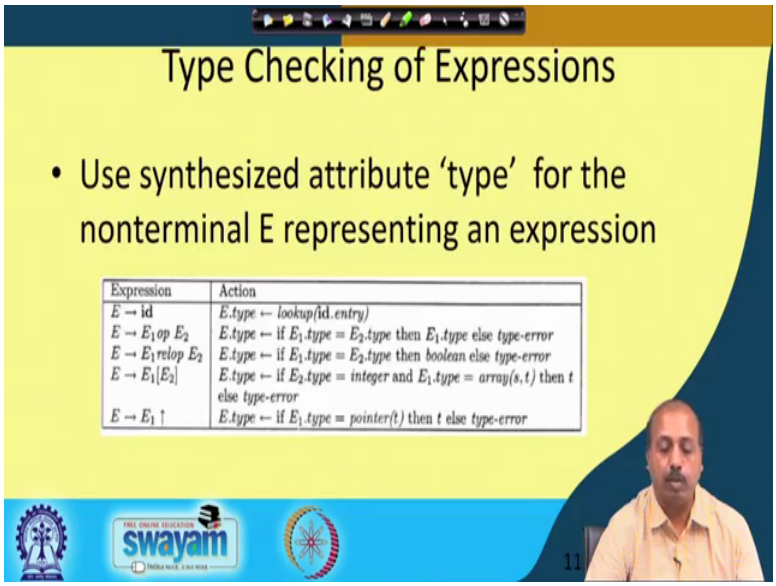
type point of view that will depend on the interpretation of f 1, but in C language you know that it implicitly assumes that f 1 is of type integer. So, that way so, this does not put this put this requirement that f 1 has to be defined previously and things like that. So, there is this type of this type of flexibilities may be there as a result we can have this weakly typed constructs. So, weekly type things so, it will it will make the life of the programmer easy, but at the same time it also gives rise to the problem that sometimes the program may generate some error while executing actually in actual.

So, strongly typed languages they have got put a lot of restrictions and there are cases in which a type error can be caught dynamically only. As I was telling that say some array bound increasing beyond the limit; so, there those type of checks cannot be done statically. So, they are to be done dynamically only and similarly when you are say calling a function pointer. So, when it is returns returning a function. So, whether that function is of proper type or not, so, that can be checked only dynamically because it will be decided at that point only whether the function that has been returned is returning a proper value, ok, it is returning a proper value of correct type for assignment to some other variable. So, that way it can it can be checked dynamically only.

And, many languages will allow users to override the system like; this type overriding, like type coercion and all where you can have a typecasting ok. So, for example, C language, so, that allows to this typecasting type of facility. So, it can allow you to override the system the type system.

So, these are the some of the concerns for the strongly typed language which we not only it may or it may not be followed very rigorously because of these problems.

(Refer Slide Time: 19:30)



Now, how do we do type checking for expressions as I was telling that it will be using some syntax directed translation mechanism for doing this type check. So, this so, it is so, we assume that with the non-terminals and terminals so, we will have some type, ok. So, with there is an I attribute type which will be used for representing the type of a of an expression E. So, this E producing id. So, this E dot type the corresponding action so, when this particular reduction will be done, so, what we do in the simple in the symbol table will look up for this id dot entry. So, it will be finding the entry in the symbol table and once it finds it, so, the corresponding type is also known. So, this expressions type is made equal to type of that particular entry.

Then, we have we have got a rule reduction like this E producing E 1 operator E 2. So, this operator may be any arithmetic or logic operator, ok. So, this E 1 of E 2, so, this E dot type is if E 1 dot type and E 2 dot type are same then it is E 1 dot type else it is type error. Now, this gives rise to many issues like we are so, it is expecting that the type of these 2 operands of this particular operator they are same. Of course, we have not checked whether this particular operator is applicable on these operands or not. So, that check can also be introduced.

So, if we do all those things then; so, if for example, if I am talking about say addition and this E 1, E 2 they appear to be say string variable, then this addition does not have any meaning in them. So, though E 1 dot type and E 2 type both are string so, we so, in

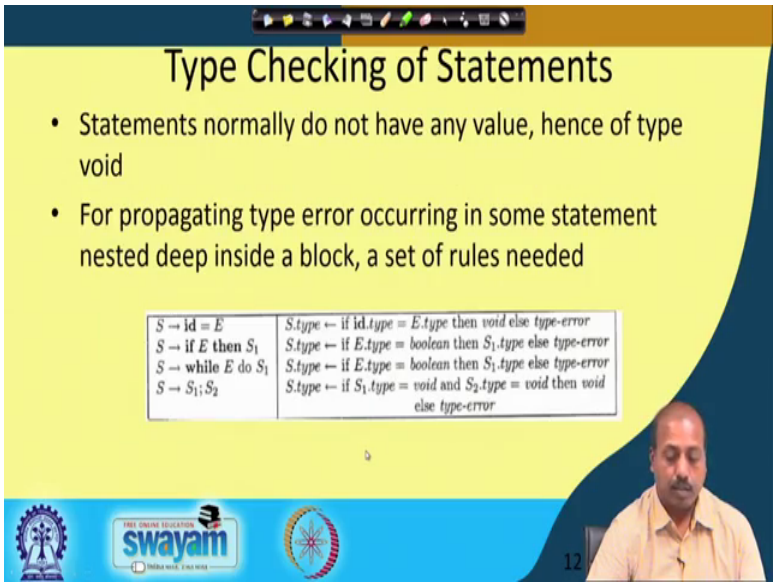
that case also the type of E should be type error. Of course, it is not captured in the rule that we have specified here. So, here it is simply checking that types of E 1 and E 2 and if there are types are same then it is said to be equal to E is type is said to be equal to E 1 dot type otherwise it is said to be a type error.

Similarly E 1 a relational operator E 2. So, say some x greater than y this type of expressions. So, here also this E 1 dot type and E 2 are type they must be same. However, if they are same after this relational operation the return type will be a Boolean variable. So, then in that case it returns a Boolean otherwise it is a type error. So, this way I can have this E 1 dot type equal to E 2 dot type then it will then it will have this Boolean type a Boolean type otherwise it is a type error.

Then this is this is for the array. So, E producing E 1 within bracket E 2. So, this E 1 and E 1 is supposed to be some name it is supposed to be of type array and this E 2 is some index, ok. So, then it is the rule that we have is if E 2 dot type is integer, so, if this is this is supposed to be an index. So, this in this must be the it is type must be an integer. So, if E 2 type is integer and E 1's type is array S T, ok. So, if E 1's type is array S T, so, this is an array and then S is the size and T is the type of individual elements in the array S T declaration. Then this in that case T will be in that case this x type of this E will be T otherwise this is a type error. So, this way we can derive the types of array elements and in an expression.

Then this is E 1 pointer. So, if I if I equal to E 1 pointer, so, in that case E dot type is so, it is assigned to be T if E 1's type is a point at to T. So, T is the basic element and if it is E 1's type is pointer to T then E's type will be T otherwise this is a type error. So, this is just a simple example of this type rules, but you can think about more rigorous rules as I was talking about. So, whether these operators are applicable, whether this arrays that we have, so, they are giving rise to further indices or not, so, multi dimensional arrays and all. So, they are not taken care of in this particular expression.

(Refer Slide Time: 24:15)



Then for the statements: so, I said in some previously that statements also can have some type, ok. So, statements normally do not have any value hence their type is void. However, so, if there is a type error in statement which is nested deep inside a block a set of rules are needed because if there is a type error in inside a block then there is no point trying to generate code for the remaining part of the program. So, that entire block can be marked that there is a type error.

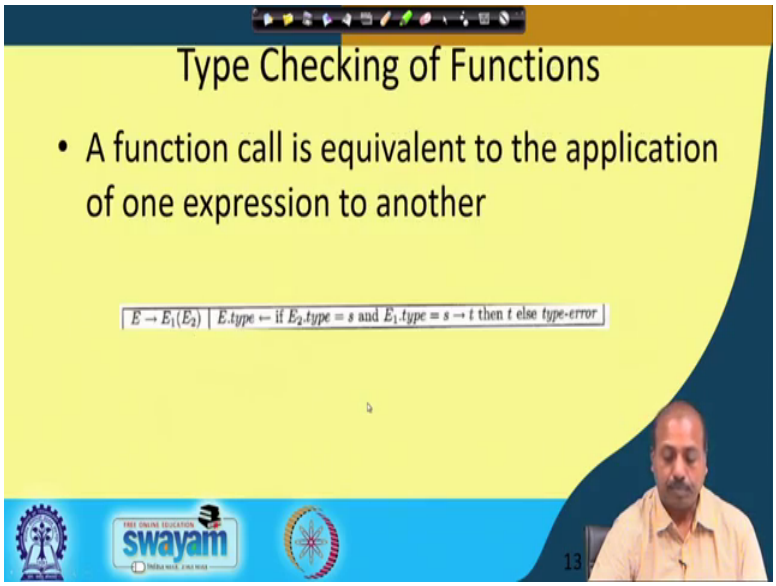
So, this is what is done like say id is producing id equal to E, so, this id type if ids type is equal to E S type in that case this is type of S is void otherwise it is a type error. So, if there is a type mismatch between id and E then there is a type error and that type error is assigned to S, so that at a higher level we can understand that there is there was a type error that has occurred with this statement. So, we can take some appropriate action. So, it may be aborting the aborting the code generation or whatever, but we can do that.

Similarly, this one so, if E then S 1. So, this S dot type so, if it is type is equal to if a dot type is Boolean then it is S 1 type else type error. So, in this so, the more many programming languages they will require that this is E's type be Boolean some other language. So, it may not require so, like for example, in C language it is E dot type need not be Boolean it has to be some it may be any integer or real any other type, but. So, here it is taken as E dot type is Boolean. Some other programming language they will require that a the expression in the if statement to be Boolean type. So, it will check that

if a dot type is Boolean then S type is equal to S 1 type. So, if the S 1 type is void then S type will also be void; if S 1 type is a type error then S type will also be a type error, but if it is not Boolean in that case so, this S type is said to be equal to type error.

Then this while statement this is also similar if E dot type equal to Boolean then S 1 dot type else type error. So, here also it is same. And, then this concatenation of statements S 1 semicolon S 2 so, occurrence of successive occurrence of 2 statements S 1 and S 2. So, if S 1 type is void and S 2 type is void then type of S will also be void otherwise type of S is type error. So, if there is an error in S 1 or S 2 then there types are not void. So, in that case it will be a type error.

(Refer Slide Time: 27:22)

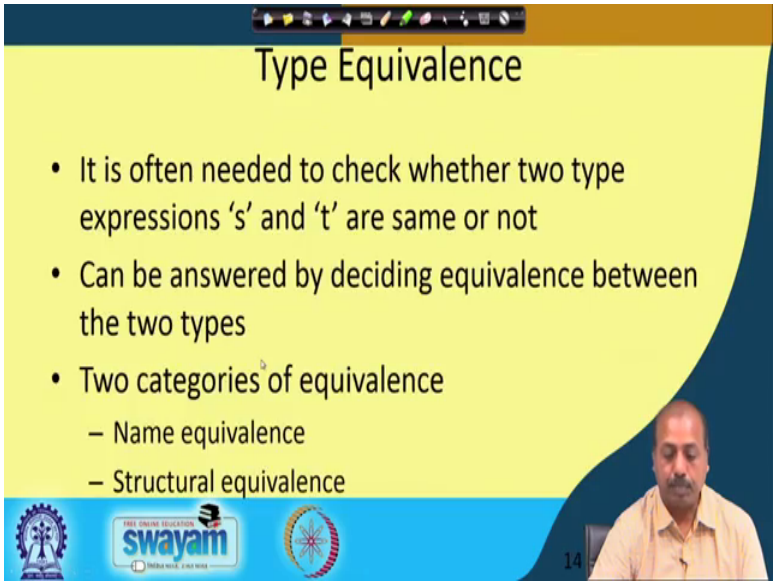


So, if type taking of functions sso, if we have got something like this like E producing E 1 within bracket E 2. So, this is E 1 is the it is a function call. So, E 1 is expected to be some id or so. And, then this E 2; so, this E 2 is expected to be the parameters that we are passing.

So, so, this is like E dot type is if E 2 dot type equal to s, and E dot type is E 1 dot type is s 2 t. So, E 2 dot types is s and E 1 dot type should be from that domain like if E 2 is an integer and then this function for E 1 it should its domain should be integer and if its domain is same as u E 2's type and the then we have to look into the range of E 1. So, range of E 1 is the return value type of E 1. So, that is t since the function given is s 2 t, so, that is the return type of it and then it will be in that case the E's type will be t

otherwise it will be a type error. So, in this way we can have type checking for functions noted.

(Refer Slide Time: 28:37)



So, we can talk about type equivalency. Now, it is sometimes it is needed to check that whether 2 types s and t they are same or not. So, we can answer this by deciding equivalency between the 2 types and this equivalency there can be 2 categories; one is called name equivalency, another is called structural equivalency. So, equivalency check is necessary because many times we need to derive for an expression like say if x equal to say y; now this x and y they may be apparently of different types, but we want to see whether they are they are of some equivalent type or not.

So, that way so, this so, if it is one type of equivalency like name equivalency so, it will it will find that if the names of the 2 types are different then it is they are not named equivalent, so, in that case it is a type error. And, in some cases so, we will look for further. So, we look into the structural part of it and then we will see that whether they are structurally equivalent or not and if they are structurally equivalent then we can say that the 2 types are equivalent.

So, this type equivalency is very important because we can so, many times for the programmers ease, so, they define different types and they use different constructs for defining the types and then while combining them while checking them in the later part of the program we are we do not make the programmer constrained by the type

definitions. So, if they are equivalent type then it should be taken into consideration. So, that gives us to this equivalency.

So, we will see this equivalency rules, how to do all these checks in the next class.