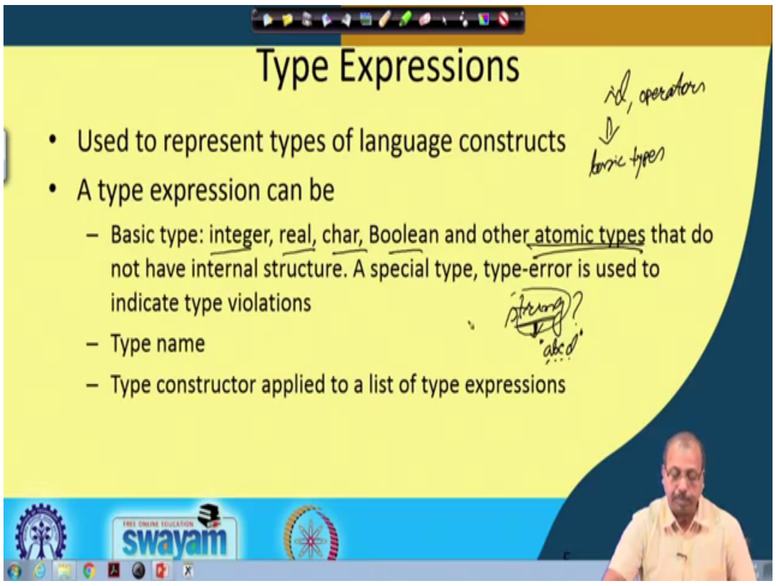
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 40
Type Checking (Contd.)

Next we will be talking about type expressions. So, type expression as the name suggests. So, this is an expression of types ok.

(Refer Slide Time: 00:23)



Type Expressions

- Used to represent types of language constructs
- A type expression can be
  - Basic type: integer, real, char, Boolean and other atomic types that do not have internal structure. A special type, type-error is used to indicate type violations
  - Type name
  - Type constructor applied to a list of type expressions

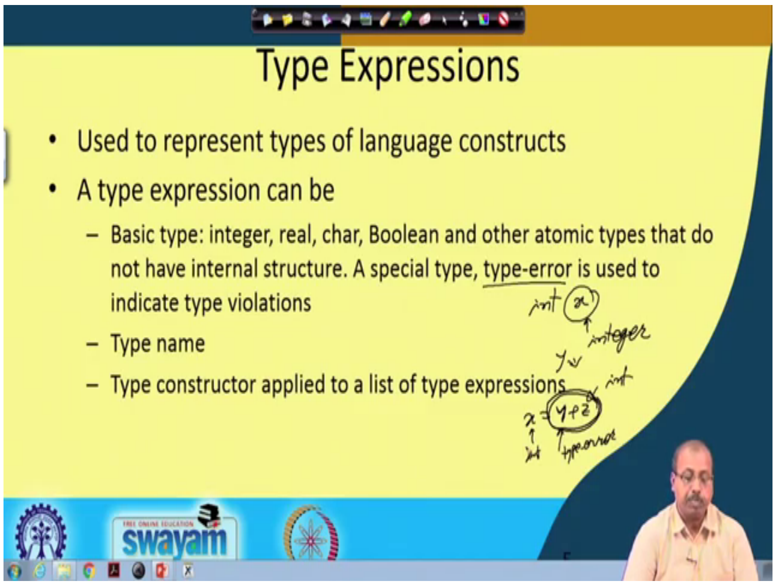
So, instead of it is being the type of expressions, so, I will read it as expression is the involving types. So, it is used to represent types of language construct. So, we will make an expression whose individual elements will be some types and they will be utilized for representing types of different programming language constructs. So, type expression. So, it can have basic type like any expression can have the basic ids identifiers and some operands and all. So, like here also for type expression, so there can be some basic types which are similar to which are synonymous to identifiers in a normal expression.

So, if I have got a normal expression. So, there are some id identifiers are there and some operators are there. So, in case of type expressions so these id's are basically the types themselves. So, the basic types the basic types. So, they will be the id. So, basic type like integer, real, character, Boolean and other atomic types that do not have internal structure. So, like a programming language. So, that can define like what are the basic types that are supported by it and they so maybe some programming language will support beat as a type. So, that the basic types of those types which cannot be broken down further they are atomic you can say. So, they are you cannot break an integer into further types ok. So, that is the thing. So, then if it is the case then that will be called a basic type.

So, now, what is the basic type? So, for example, what about say string. So, if I say a string. So, is it a basic type now the answer depends on the programming language like somebody may say yes string is not a basic type because I can break this string into characters. So, this a, b, c, d if it is a if it is a string. So, I have got this a, b, c, d as individual characters in it. So, I can see that there are characters inside it, but seeing that there are characters inside it does not say that this is a this is this is not a basic type. Why am I saying? So, like given a string. So, if there is an operator by which I can extract the individual characters and assign it to the I can access this individual characters as characters only. So, if I can do that in that case I will say that string is not a basic type. But if it is such that on string you can only do operations like screen concatenation, string copy, then string reversal and all that. So, only those type of operates are allowed, but you cannot extract the individual characters for from the string. So, if you cannot do that. So, in that case string also happens to be basic type.

So, that is what I want to mean is that the definition of basic type is more on this atomicity behavior. So, in some if you cannot break the type further then I will call it as on as a basic type. So, this integer most of the programming languages they will be using this basic type. And there is another special type which we will call type error. So, which is used to indicate type violation. So, if I have got an expression of type. So, what is its type?

(Refer Slide Time: 04:05)



Type Expressions

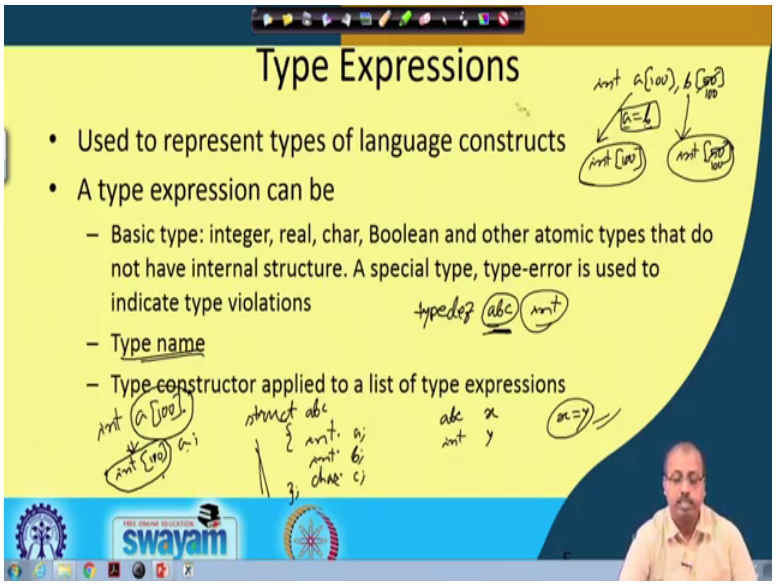
- Used to represent types of language constructs
- A type expression can be
  - Basic type: integer, real, char, Boolean and other atomic types that do not have internal structure. A special type, type-error is used to indicate type violations
  - Type name
  - Type constructor applied to a list of type expressions

So, suppose I have got say an integer. So, I say that int x. So, when I am trying to derive the type of this x. So, I am telling its type is integer fine. Now if it at some place I suppose there is a variable y whose type is not known, then its type is unknown and if I am trying to do an operation likes the x equal to y plus z then what is the type of this expression y plus z. So, y plus z I cannot do because I do not know the type of y maybe z is of type integer, but for type y I do not know what is the type. So, in that case type of this whole expression will be marked as type error.

So, that you can so it is not confused to be either integer or something like that, but my process can continue, my compilation process can continue. So, when I am trying to compare this type of x with the type of this expression I see that this expression is a type error and this a x is of type integer. So, naturally this assignment cannot be done. So, there is a type mismatch. So, the compiler designer can detect this situation. So, this time period is a another basic type; so, apart from the basic types like integer, real, character, Boolean etcetera. So, we will also have type error as another basic type.

So, it can be the name of a type. So, maybe we do define a new type and then that is a type name. So, I define that say for many programming languages they will allow this type def.

(Refer Slide Time: 05:45)



Type Expressions

- Used to represent types of language constructs
- A type expression can be
  - Basic type: integer, real, char, Boolean and other atomic types that do not have internal structure. A special type, type-error is used to indicate type violations
  - Type name
  - Type constructor applied to a list of type expressions

So, type def abc integer. So, this means that I am defining a new type whose name is abc and whose values contained will be the value that values will be same as the type integer. So, this so, this is not this is a type name. So, I am giving in the type name abc whose type is actually which is actually integer, but I can define variable x to be of type abc and I can have another variable y of type integer. Now when I am writing x equal to y whether it is an error or not whether this is a type error or not that is a question.

So, some programming language may be very strictly typed. So, it will say yes it is a type mismatch because abc x is a type abc and y is a type integer. So, the strongly typed languages so, they will say this is a type error whereas, for the languages which are more flexible. So, it will try to look through for look further and see that they are derived from the same basic type integer. So, as a result this assignment maybe through. So, it is programming language dependent. So, that is the type name.

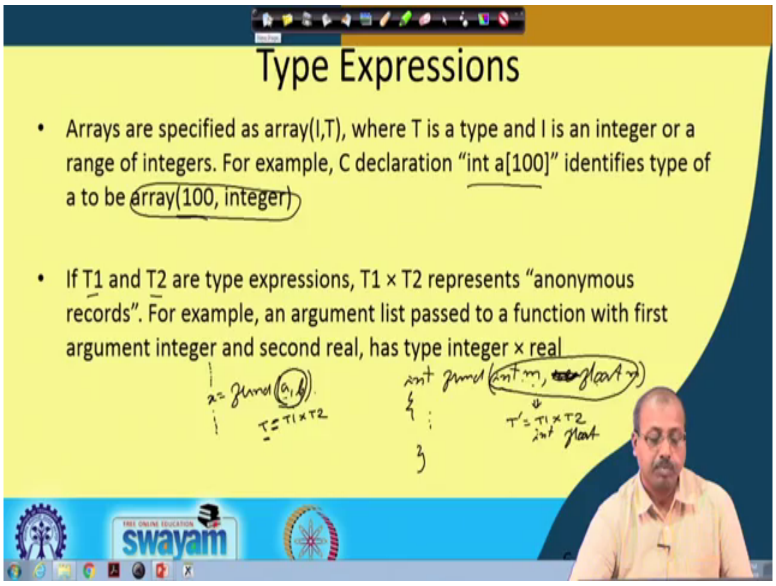
And type constructor applied to a list of type expressions. So, list of type expressions like I can say many programming languages like say you can have this structure definition. So, structure abc and there you can have different types so, integer some field maybe they are integer a, integer b, characters c. So, like that we are defining as types. So, this so, this is a constructor. So, a list of types. So, it is collection of one integer, two integer and one character. So, integer, integer and characters they are combined together to it is a list of type expressions are taken together to get this constructor structure.

Similarly, I can have say integer a 100. So, when I am saying this then I am telling that a 100 is a new type where which is our array of integers. So, array of integers this whole thing is a new type like. So, this actually you should read it like this, this expression though when most of the programming languages we read it like this, but in reality it is like this integer 100 and that is a, as if this integer 100 is the type. So, array of 100 integers taken together and I am defining a variable a of this time. So, you should read it in this fashion, though for the sake of our understanding we write in this form, but it is like this.

So, similarly if I have got something like this. So, say integer a 100 and b 50. Now you see that can I write that a equal to b. I cannot because there is a type mismatch here what is the type mismatch. So, actually this a is of type integer 100. So, this is the type of a and the type of b is actually integer 50. So, it is b is a variable which is of type which is collection of 50 integers and as a variable of type which is a collection of 100 integers. So, this is also a constructor. So, this is an array constructor we can say. So, this is constructing an array of 100 integers this is constructing a type of array of 50 integers and this a and b so, they are variables of these individual types. So, we cannot do the assignment. So, the it will be stopping as at type error.

Now, some pro so, if they are same type like if this is also say 100 instead of that. So, this is also 100 then in that case a and b both are of same type. So, this is also 100 of both of them are of type of integer 100. So, in that case a equal to b is a feasible operation. So, many programming languages do allow this array assignment directly. So, it will be doing element to element allocation. So, element to element copy, but it is allowed in the some programming languages so, but this type has to be same. So, this way this type becomes very critical. So, it is the just by looking into the program, so, we cannot say that this is they are there this types are correct or not. So, it is very much defined by the underlying programming language that semantics like whatever is given there.

(Refer Slide Time: 10:51)



Type Expressions

- Arrays are specified as array(I,T), where T is a type and I is an integer or a range of integers. For example, C declaration "int a[100]" identifies type of a to be array(100, integer)
- If T1 and T2 are type expressions, T1 × T2 represents "anonymous records". For example, an argument list passed to a function with first argument integer and second real, has type integer × real

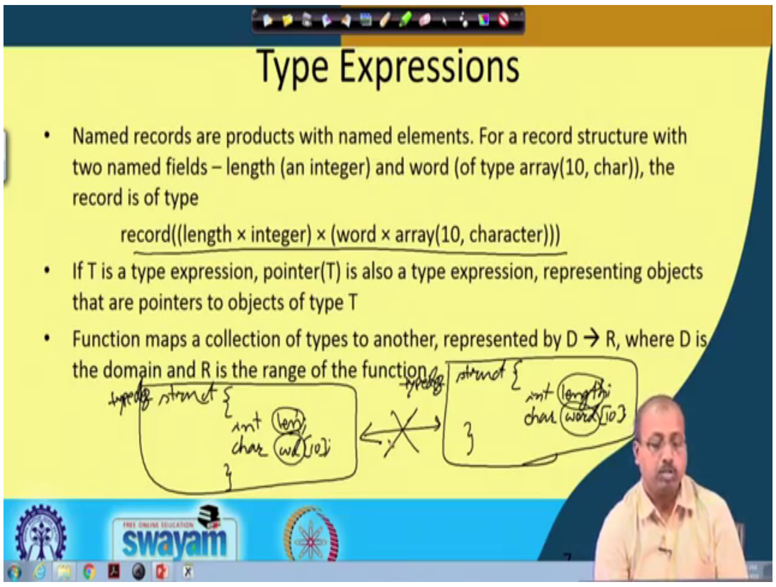
So, next we will be looking into this thing that. So, arrays are specified as array I, T where T is the type and I is an integer or a range of integer. For example, so, this is basically said that is what I was telling that integer a 100 is nothing, but integer a 100 is nothing, but is a type of a is array 100 integer. So, it is written like this array it is a constructor array and hundreds of integers are taken together to give this thing.

Now, we have got this thing that if T1 and T2 they are two type expressions then this T1 cross T2 it represents anonymous records. So, for example, when you are passing an argument list to be a function the first argument integer a second real has type integer cross real. So, what I mean is suppose I have got a program and at this point I am calling the function say function 1 and there I am passing two values a and b and here in the function the function is like this. So, here int m and say float n and there I have written the function. Now I need to check that when I am calling this function func 1 from this point. So, the parameters that I am passing is really matching with the argument, the arguments that I am passing here is really matching with the parameters that I have here in the function. Now how do I do this thing?

So, actually the way we are doing is we are taking these a matching if this a with the first argument first parameter matching the type of b with the second parameter like that, but formally speaking. So, what are we doing? So, the, if I say that the for this is a type expression. So, then this type expression is given by of T1 cross T2 where T1 is the type of the first parameter and the first argument and T2 is the type of the second argument as a result this gives me a new type which is the cross product of these two types T1 and T2 and then and then what is the type of this. So, this is nothing, but a type T dash which is again T1 cross T2 and then this T1 and T2. So, this is this is integer and this is float.

So, T dash that we have here is a type integer cross float. So, here also this T should must be integer cross float then only I can say that these two types are matching otherwise not. So, this way formerly this parameter passing technique that we have that is also we have to check for the arguments and we do this thing. So, some programming languages where it is where you have to do this type checking. So, they will do it like this some programming languages will say no I am free. So, you can pass any arbitrary arguments. So, that is that is possible. So, then this checking is not done. So, this type construction is also not done, but anyway. So, for most of the programming languages, so it will do this type checking and this will be required.

(Refer Slide Time: 14:31)



Type Expressions

- Named records are products with named elements. For a record structure with two named fields – length (an integer) and word (of type array(10, char)), the record is of type

  record((length × integer) × (word × array(10, character)))

- If T is a type expression, pointer(T) is also a type expression, representing objects that are pointers to objects of type T
- Function maps a collection of types to another, represented by D → R, where D is the domain and R is the range of the function

```
typedef struct {
        int len;
        char wd[10];
}
```

```
typedef struct {
        int lengthi
        char word[10]
}
```

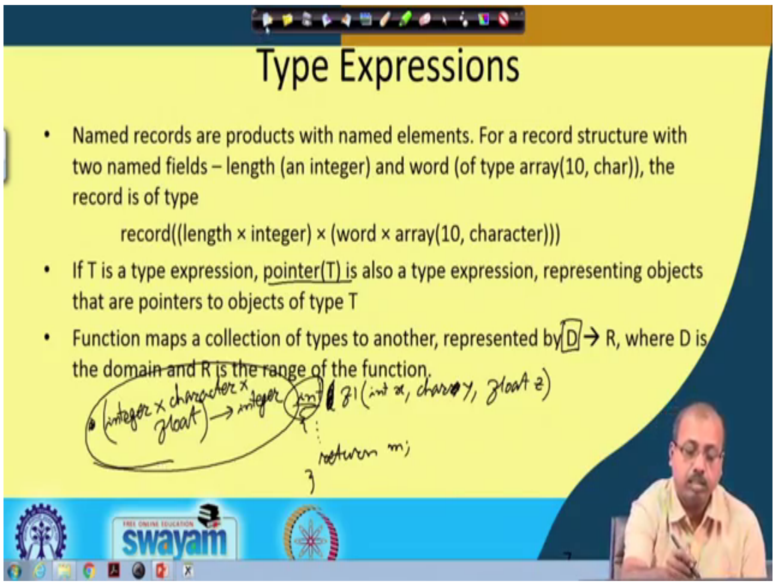
Next we will see like the named record. So, how does it record will look like. So, named record so, they are products with named elements, for a record structure with two for named fields length and word the record is of type length into integer length cross integer cross word cross array of 10 character. So, this is the this actually telling me that I have got a record say for example, if you look into the c programming language. So, this is define a structure where the first field is first field is it has got a name length and this is an integer. So, integer length and the second field is word which is of array 10. So, it is like character word 10. So, this is the structure.

Now what is the type of this structure. So, that is what we try to figure out. So, its type is depicted by this particular type expression. So, it is length cross integer. So, length is of type length is of type integer. So, length into integer because length is a name of the field and integer is its corresponding type and this is word is the length name of the field and it is an array of 10 characters so this product. Now what it means is that this if I have got another structures say structure so, integer instead of length so, I call it say len and instead of word I call it say wd.

Now, whether these two structures are these two types that same or not. So, this is also a type def, this is also a type def in from c language. So, they both of them are defining types. So, are these two types same or not. Now as far as this definition is concerned this types are not same because here it says that it should be length cross some integer value, but here it is len cross some integer value. So, they are not same. Similarly this is word cross array 10 character, but here it is wd cross array 10 cross characters. So, 10 character. So, naturally so, they are not equivalent if I go by this thing. But of course, if you go further so, you can see they can say that this len and length they are actually equivalent because they are both of them are integers and then this wd and word so, they are also equivalent because both of them are character is obtained entries.

So, that way I can say that they are equivalent, but it requires more work ok. So, the either type checker has to do this type of checking then the work involved is more? So in most of the cases what happens is that we stop at this level itself telling that these two types are not equivalent ok. So, we will come to this when you go to the type checking in more detail. So, this is the named record how are they how they are types are derived so, that is mentioned here. And also you can have a pointer. So, pointer the if T is a type expression, then pointer T is also a type expression that represents objects that are pointers to objects of type T.

(Refer Slide Time: 18:25)



Type Expressions

- Named records are products with named elements. For a record structure with two named fields – length (an integer) and word (of type array(10, char)), the record is of type

  record((length × integer) × (word × array(10, character)))

- If T is a type expression, pointer(T) is also a type expression, representing objects that are pointers to objects of type T
- Function maps a collection of types to another, represented by $\boxed{D} \rightarrow$ R, where D is the domain and R is the range of the function.

(integer × character × float) → integer

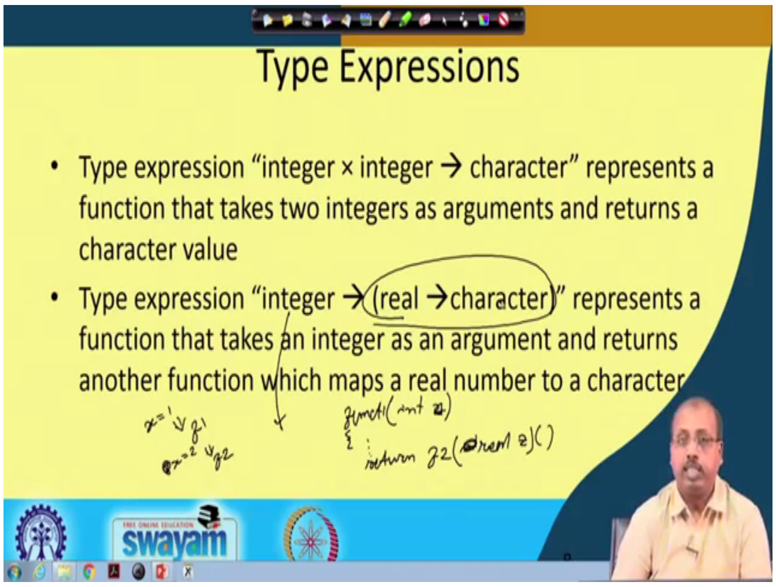
```
int f1(int x, char y, float z)
{
  return m;
}
```

So, this pointer T is also this is also a type expression whose typing. So, this will represent objects that are pointer to objects or type T and a function maps a collection of types to another representing D to R where D is the domain and R is the range of the function.

So, for example, if I have got a c function like integer say f 1 is the name of the function and it takes parameters like say integer x, character y, float z then it. So, finally, returns some into your value as the type of the f 1 is integer. So, it returns some integer say m fine then D is the domain. So domain says that from which set it can pick up the values. So, it can pick up the values. So, from that domain which is integer, integer cross character cross integer cross float. So, this is the domain because it can take up values from this domain. So, in the how is this domain? So, these domain is collection of three entries where the first entry is an integer, second entry is a character and a third entry is a float ok. So, this is D and this range is the output of the function the output type is integer. So, you can say that given three values of this type one is the one integer one character and one float the or the rather the first one integers, second one character, third one float is ordering really matters. So, it will be outputting an integer.

So, this is the type of the function. So, if I tell what is the type of the function so, type of the function is this thing. So, it takes parameters integer will first parameter integers, second parameter character, third parameter float and it outputs one integer value. So, that way so, function maps a collection of types to another represented by D to R where D is the domain and R is the range of the function. So, we can we can represent these functions like that.

(Refer Slide Time: 21:07)



Type Expressions

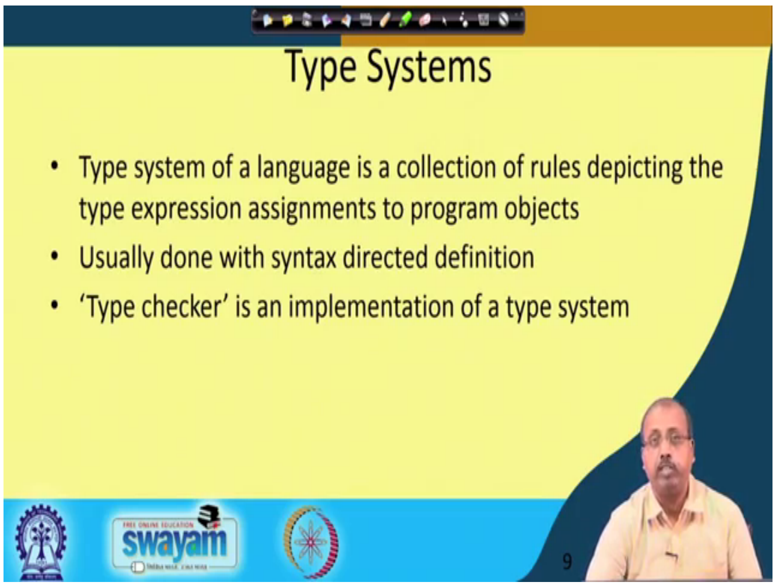
- Type expression "integer × integer → character" represents a function that takes two integers as arguments and returns a character value
- Type expression "integer → (real → character)" represents a function that takes an integer as an argument and returns another function which maps a real number to a character

Now, once we do this then the next one. So, for; so, this is another example like the integer cross integer to character. So, this type expression it will represents a function that takes two integers as arguments and returns a character value. So, that is one possibility then the type expression integer to real to character. So, this represents a function that takes an integer as an argument and returns another function which maps a real number two a character. So, this is like this that so, either if I look into this one more carefully. So, this is basically that function pointer type of concept that you have in many programming languages like c.

So, what the function does is the function 1 so, it takes an integer value integer x and depending on the value of x it will return some other function return some other function f 2 so, where which will be taking a real number and it will be returning the characters. So, this returns the so, it will be; so, it will return the function f2 such that it takes some real number say it will take some real number and this f 2 will be. So, this is often written like this to tell that f2 is a function pointer. So, like that. So, this is a point function pointer. So, it depends the syntax depends on the programming language that we are looking into, but essentially what I mean is that given the value of x it will return some function maybe if x equal to 1 it will return the function f1, if x equal to 2 it will return the function f 2. So, like that as a return value it does not return a single value, but it returns another function and that function that it is returning is of type like this. So, this function f1 given a real value it will produce a character or the function f2 given a real value it will produce a character.

So, this way this function pointers can also be taken care of in the type expression. So, this of course, makes it bit complex, but it can be taken care of there.

(Refer Slide Time: 23:32)



Type Systems

- Type system of a language is a collection of rules depicting the type expression assignments to program objects
- Usually done with syntax directed definition
- 'Type checker' is an implementation of a type system

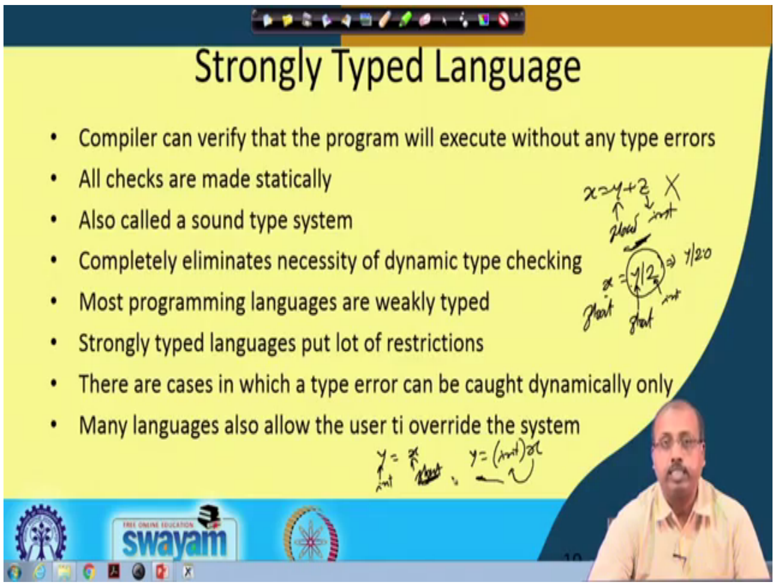
Now, what is a type system? So, type system of a language is a collection of rules depicting the type expression assignments to program objects. So, so, if you look into the programming language manual then apart from giving the grammar that has got how this individual constructs of the grammar will be will be there how are they organized, what is the grammar for that. So, will also tell you like how what are the types which types are allowed and all so, it will give you all those details.

So, that gives rise to a type system. So, type system it is a collection of rules that will depict the type expression assignment to assignments to program objects and usually done with syntax directed definition. So, this type checking and type conversion. So, they are often done in the syntax directed translation phase itself then we will see how this can be done and type checker is an implementation of the type system. So, as a compiler designer so, we also have to include a type checker. So, it is not only that we generate code or do some syntax directed translation to do something else. So, that is there, but apart from that.

So, you also need to have some type checking rules and those type checking rules are also to be made part of this syntax directed translation phase. So, that will be known as type checker.

So, this is more difficult if the language is strongly typed them this type checker will be more complex the language is relatively simple then the type checker will also be a bit simple.

(Refer Slide Time: 25:12)



Strongly Typed Language

- Compiler can verify that the program will execute without any type errors
- All checks are made statically
- Also called a sound type system
- Completely eliminates necessity of dynamic type checking
- Most programming languages are weakly typed
- Strongly typed languages put lot of restrictions
- There are cases in which a type error can be caught dynamically only
- Many languages also allow the user ti override the system

So, what is a strongly typed language? So, strongly typed language, so, compiler can verify that the problem will execute without any time errors. So, this is the most important like the compiler. So, if the strongly typed language. So, if the compiler says yes the program is correct syntactically and semantically correct then, so, it will be it is telling that the program will not give those type of bugs, those type of problems while executing it will not come up with some type error.

So, it is not that there is an assignment like x equal to y and while executing so, it will be coming up with a statement that there is a type mismatch and that is why there is a there is an error in the program of course, the logical bugs maybe there in the program that cannot be captured by any compiler, but so, type related issues are there so that can be detected by the compiler itself and once the compiler certifies that the program is correct from the type point of view. So, there cannot be any problem during execution.

All checks are made statically. So, they are before the program is run and before the code is generated all the checks are done and this is also called a sound type system because, because of the strongly typed, so everything is known everything is fixed. So, this is called a sound type system. So, it completely eliminates the necessity of dynamic type checking. So, of course, so, that dynamic type checking will not be necessary because everything will be checked statically of course, at array bounds check that we have talked about.

So, it cannot be done at the compilation time or the statically, but if that array bounds check is not there then of course, I do not need to do it dynamically. So, if the language does not require this array bounds check then h I can say that it will be it does not require an dynamic type checking.

Most programming languages are weekly typed, they are they are not strongly typed because strongly typed means for every small things. So, we have to you have to have this type things taken care of. So, the programmers will also find that many things are too trivial and this too trivial things are also not taken care of like that may be the situation; so, like so, that creates a difficulty. So, that is why most programming languages they are weekly type and this is strong strongly typed languages put a lot of restrictions on the programmer.

So, we will see that if there are a lot of restrictions so the strongly typed language so it will it will not allow you to do arbitrary assignment and operations. So, as a result it is like that for example, so, if I have got say this x equal to y plus z then this is y is of type float and z is of type integer. So, strongly typed language you will not allow you to do this addition, but you see that we can do this addition because we can always take it as a floating point addition and do the addition.

So, if it is a strongly typed language then everything has to be everything has to be very crucial. Another very subtle example maybe like this, suppose I want to write like x equal to y by 2 where x and y both of them are of type float, but while I am writing like this y by 2 this 2 is it 2 is an integer and this why is it floating point number. So, you see now there is a type mismatch because the two operands involved in the division operation they are not have same type. So, it requires that the programmer writes it as y by 2.0 not as 2. So, that these type of funny things can come up. So, that is why if you make it very strongly typed. So, it can give rise to many problems for the programmer also. So, this strongly typed languages they are; they are not this type checking does not is not made so strong. So, there are cases where this type error can be caught dynamically only.

So, this is a typical situation is like that array bounds check that I was talking about and many languages also allow the users to override the system. So, like at the type casting type caution. So, those are there. So, we will see that in subsequent classes. So, there we can override this type system and say the programmer says that no these type assignment is correct. For example, in c language so if I have got this thing so, y equal to say this x and x and y are of two different types then says maybe x is say integer and y is say float.

So, we often write like y equal to int x ok. So, that this will be overriding the rule that y the integer cannot get the value of a float. So, if you do this; that means, is x is converted to integer and then only it is assign. So, this type of overriding may be possible in the strongly typed language whether this many programming languages they will bypass this strongly typed concepts by having these extra things in the language itself.