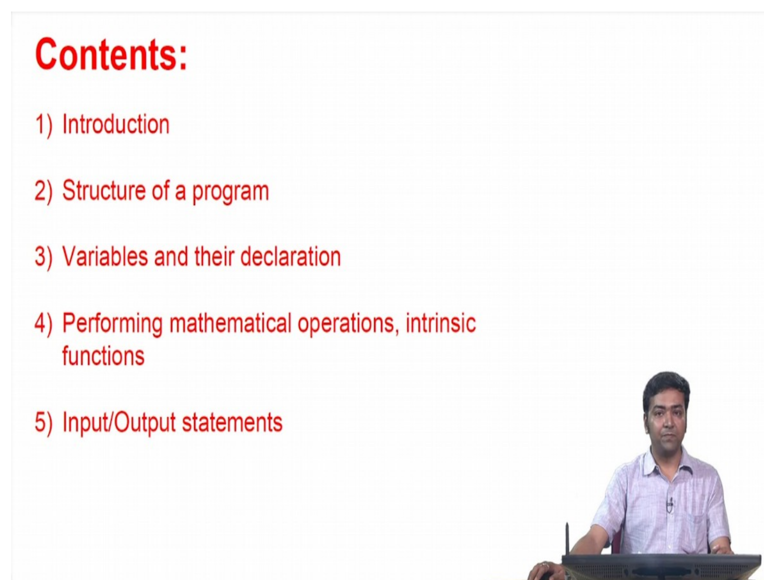
Computational Physics
Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture - 01
Introduction To Fortran Part-1
Session-A

Welcome to the first module of this course which is titled Introduction to Fortran. So, the purpose of this module is to give you a brief idea about one of the commonly used programming language in the scientific community and it's also a old language, but before I go into the details I would just like to state that this is not a course on Fortran.

So, we will just quickly brush through the basic parts. So, that at the end of this module you are able to write a program on your own. So, this will be particularly useful for people who have not been exposed to a formal programming course. So, this whole module I have divided into three parts.

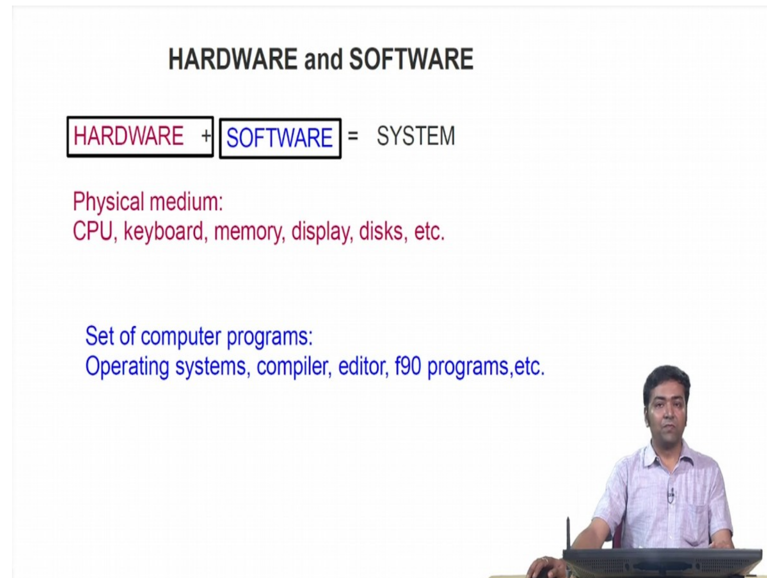
(Refer Slide Time: 01:03)



So, the first part; the contents are as follows. So, I will provide a brief introduction to the Fortran programming then I will talk about what a typical structure of a Fortran program look like. Then I will discuss about how variables are declared inside of Fortran program.

Then how one performs mathematical operations using intrinsic functions and other stuff and finally, the how one reads input from a file or from a screen and prints out the output or the work that the program has done onto a file or on a screen ok.

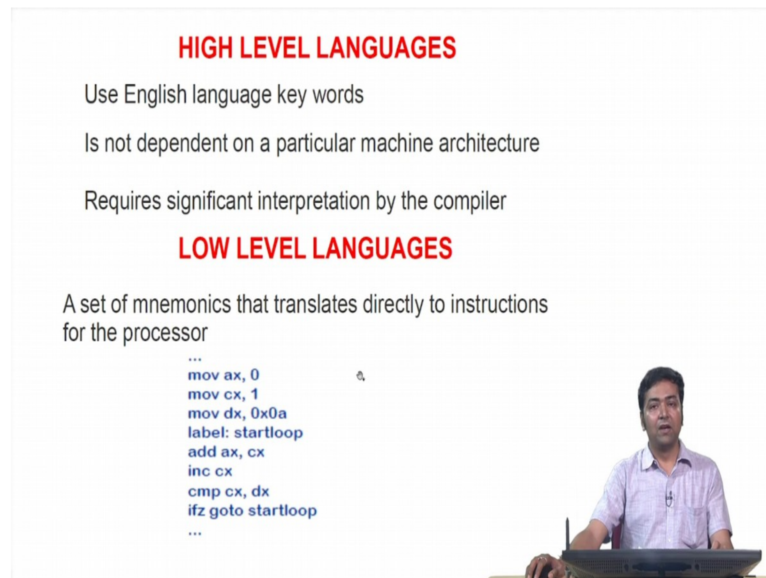
(Refer Slide Time: 01:43)



So, as of if you know that a computer consists of typically two parts a hardware which is the tangible thing and which we can see for example, a CPU, keyboard, memory, display, disks etcetera and then there is another part which is the intangible part which is the software. And together they constitute the system.

So, software consists typically a set of instructions which the machine can understand and they typically form of operating systems, compiler, editor, then different types of programs which can do the jobs.

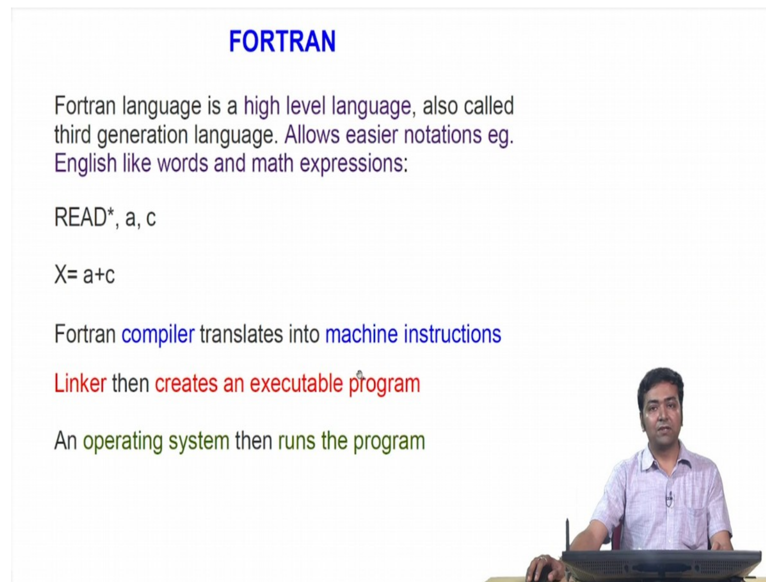
(Refer Slide Time: 02:23)



So, as the computers evolved the way a human interface interacts with the computer , that has also evolved significantly. So, in the early days so, there were the presence of these low-level languages. So, the first level language was based just on simple binary instructions, that is providing instructions to the computer by just use of binary numbers like 1 and 0.

Then the next level of language which is called the second level that is typically something which is called an assembly language and they consists of simple English alphabets or numbers, but still they cannot be easily deciphered by a programmer. So, for example, here is the use of such example of such a low level language and the language which we are going to discuss today that is Fortran that forms in the third generation language or and it forms in the category of a high level language.

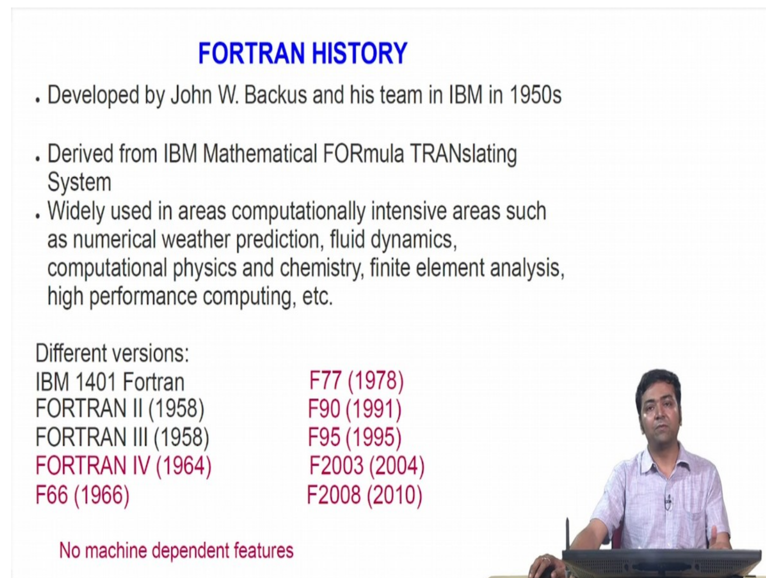
(Refer Slide Time: 03:16)



So, in this typically we use words like we use in an English language, we use math expressions as we do calculations using pen and a piece of paper. So, for example, our typical Fortran statement looks like this it has a 'read' which we can all read, it's a English word and then there are alphabets and then we have algebraic expressions like this.

So, there is a compiler which is a Fortran compiler and what it does is this compiler reads the set of instruction which the programmer writes and then converts into language which the machine can understand. So, that is basically the purpose of the Fortran compiler. So, once that is done, there is another part which is called the linker; the linker then creates an executable program which is used by the machine and the operating system to run the program.

(Refer Slide Time: 04:27)

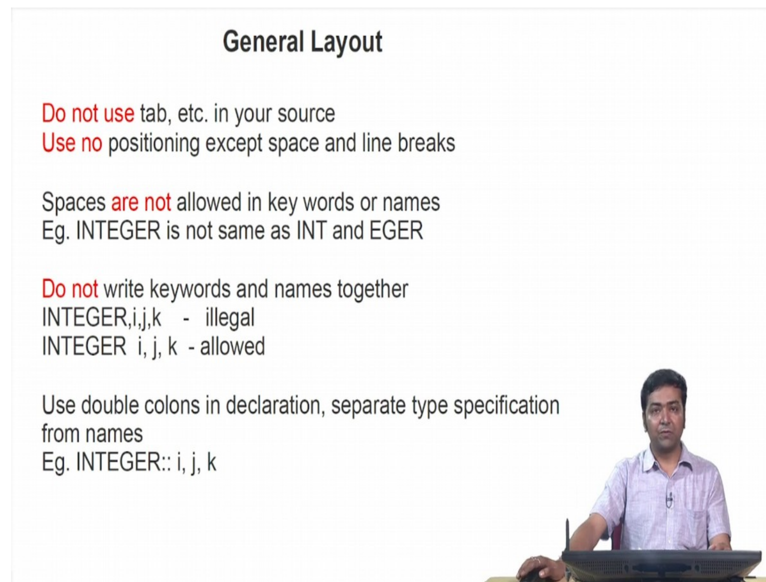


So, here I put forward of put down a brief history on a Fortran. So, Fortran was first developed by this gentleman named John W Backus and his team in IBM in 1950's after that as you can see this has evolved quite a bit and to the best of my knowledge Fortran 2008 is the last one.

The good thing is that use with this language you do not have any machine dependency. So, the same program which you write in a Linux machine say for example, using the Ubuntu operating system you can write it in another use the same program in another machine with a different operating system for example, the St. OS system.

So, that makes this thing that the program to be highly transferable. So, we can easily move from one machine to another and all we need to do is when we move from one machine to another is just recompile your program we do not need to write the program for scratch.

(Refer Slide Time: 05:30)

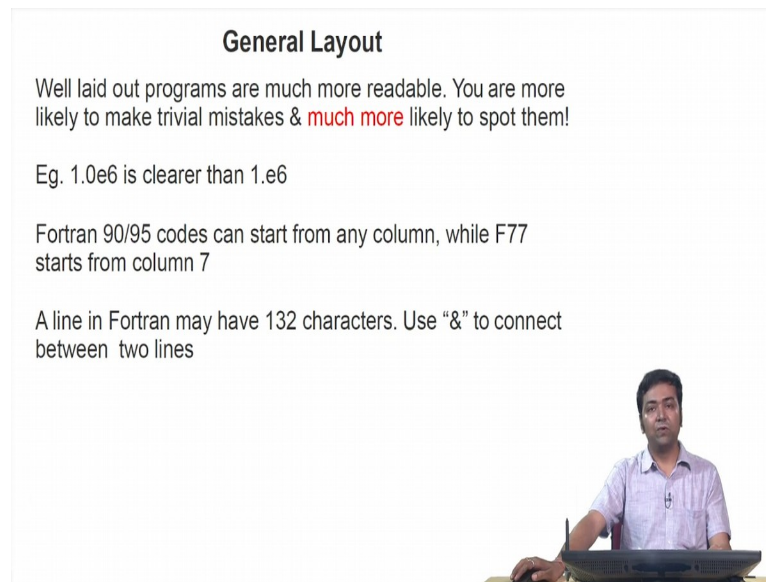


So the general layout of a Fortran program is as follows. So, for example, in Fortran we do not use tab in my source file or end of the source file, by source file I mean the file which contains the program written by me. Also when we do not use any positioning flags I accept the space and the line breaks.

Another thing we should also remember is that when we use Fortran keywords for example, in this line here the integer is a Fortran keyword ,we cannot write it in this way 'INT' , a space and 'Eger'. So, basically what it means is that spaces are not allowed in Fortran keywords also one should not write keywords and names together. For example, here I have Fortran keyword integer and then right next to it I have written the variable names so, 'i', 'j', 'k' are variables.

So, this is not the correct way to do it. So, this is an illegal way to do it, the code will complain when you compile your program and the proper way to do it is shown in this line where I write the keyword, I give a space here and then I write the variable typically in the new versions of Fortran, what one uses is double colons in the declaration and then separate type specification. For example, we have integer here then I put which is my Fortran keyword, I put two colons here and then I have the variable names 'i', 'j', 'k'. So, these colons here what they do is they separate the keywords from the variable names.

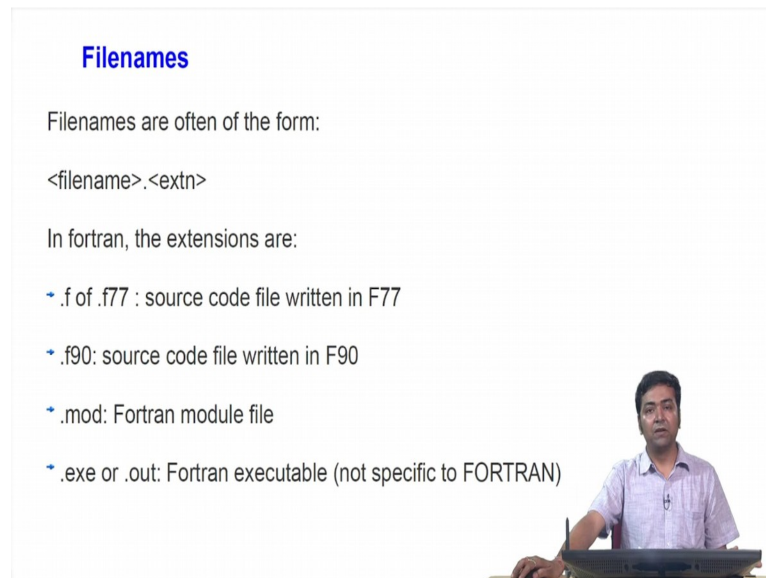
(Refer Slide Time: 07:19)



Also I mean I have listed down some good practices of typical when you write a program and this is quite independent of what language you are using. So, first of all your program should be well layout, the advantage of doing this is it makes your program more readable. So, which as a result when you later come back to your program and try to debug them, so you will more likely be able to spot mistakes in them. So, for example, a good practice is that if I write one to write 10 to the power 6. So, one way to write it is 1.0e6 another way is 1.e6.

So, I would always prefer the first one because this tells me clearly much more clearly compared to the second way of writing. So, another thing is Fortran 90 and 95 codes they can start from any column, but if someone is using a Fortran 77 the first letter in the line starts from column number 7. Line in Fortran may have 132 characters and if the number of characters exceeds that then you need to move to the next line and to connect two lines uses this percent sign between these two lines.

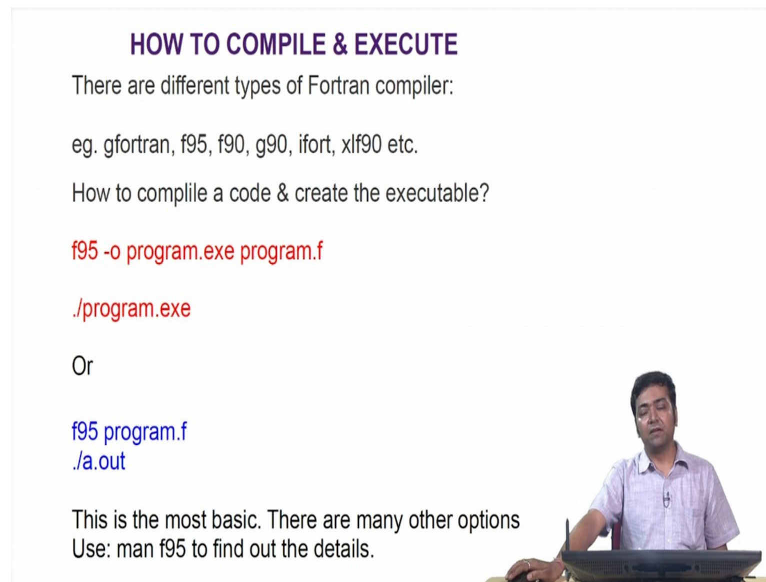
(Refer Slide Time: 08:38)



So, what I will do in the next in the remaining part of the lectureis.... Ok so, before I go there, so there is another thing which I would like to mention. So, in Fortran typically the files are named in the following way. So, you have a file name and then an extension. So, the typical extension of a Fortran program is 'dot f' or 'dot f 77' if you are writing it with a Fortran 77 compiler or 'dot f 90' if you are writing it with F 90 compiler.

Then Fortran also contains some module files which are written whose extension is denoted by 'dot mod' and then the executable find switch the computer uses to run the program these are typically have a extension of dot exe or dot out and these are not anything specific to Fortran, any executable generated by any compiler typically has these type of extensions.

(Refer Slide Time: 09:39)



So how do you compile and execute a program? So, there are different types of Fortran compilers available in the market for example, gfortran which is a g n o based Fortran compiler which is free, then you have ifort which is an Intel Fortran compiler for which you need to pay or you need to purchase from the market then you have xlf90 which is a Pascal compiler which is again not free.

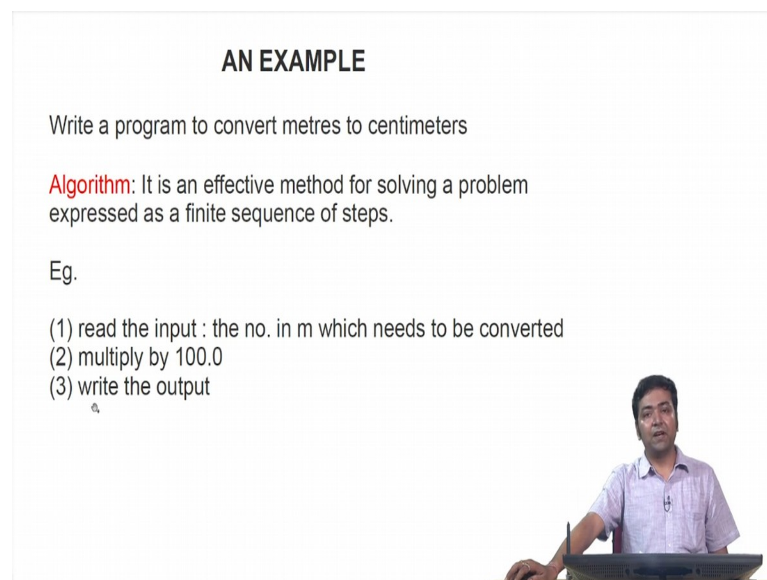
So, now the question is after I have written a code how to compile a code and then create the executable. So, that you can do it in two ways; so, one way is from the command line you type f95. So, for example, I am assuming here that I am using this Fortran 95 compiler f95 I give a space then I put minus o space program dot exe space program dot f. So, program dot f is the input file in which I have or the Fortran or the file which contains the Fortran program written by me.

So, if I type this on my command line and press enter what the compiler will do is it will compile link and produce the output of that in the executable file called ‘program dot exe’. Then to run it again from a command line you need to press you need to type ‘dot’ slash ‘program dot exe’ and then press enter. So, another way to do it is you can completely ignore this part of the line. So, that is the minus o program dot exe you can just completely ignore it you can just type from your command line f95 ‘space’ ‘program dot f’.

So, the difference between this previous one and this one is that in the second case you are not specifying any specific filename where the compiler will store the executable. So, as a result the default file name is ‘a dot out’ . So, this is what the file, the executable file the compiler will produce and then to run it you just type dot slash ‘a dot out’ press ‘enter’.

So, there are several other options and features which are available in Fortran which we are not going to discuss in this lecture. So, if someone wants to find out that several different options of compiling and all. So, one can look into the manuals which one can access by pressing typing ‘man’ space the compiler name and then you press enter it will show up in your screen or you can go to the website and try to find out ok.

(Refer Slide Time: 12:16)



So, now what we will do is we will begin by looking into how one can declare variables, how one stores memory for them, then how one performs simple mathematical instructions and then how one prints out the output. So, to do so, what we will do is we will start with an example.

So, the job I want my computer to do is I will type in a number which will be given in meters and then I want my computer to convert it to centimetre. So, how do I pass on all these instructions which I have just spoken about to the Fortran compiler and then the compiler passes it to the machine. So, this is conceptually developed by using something which is called an algorithm. So, for example and so, an algorithm is basically an

effective method for solving a problem expressed as a finite sequence of steps. For example, related to this present example which will be looking in, so an algorithm will look like something like this.

So, you read the input that is the number in meter which needs to be converted. So, to convert a number in meter to centimetre we all know that we to multiply it by 100. So, the second step is the number which has been read in needs to be multiplied by 100 and the result of that needs to be written in the output.

(Refer Slide Time: 13:46)

Logical structure of a program

(1) Start of program

(2) Reserve memory for data

(3) Write prompt to display

(4) Read the length in meters

(5) Convert the length into cm

(6) Write the length in cm

(7) End of program

So, in this slide what I have is the logical structure of a Fortran program. So, the first thing is you have to in the program you have to mention to the compiler in your input file where the program starts from and then also you have to mention where the program ends. So, once you have done this then in between you need to do steps number 2 to 6. So, the second step is you have to tell how much memory to resolve for the data. Then if you think from a user's perspective someone who is trying to use a program to do that job so, he or she needs to know when he or she needs to put in the number which is in meters and which one wants to convert to centimetres.

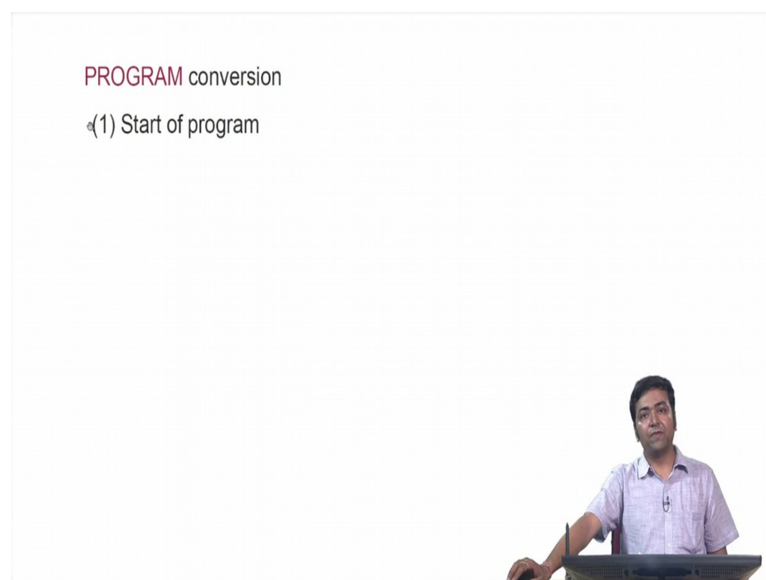
So, for that we need to write in prompt on the display; on the screen to tell the user to write the number which they want to convert and once the user has written the number then the next step the fourth one will be to read the length in meters. So, once we have the program has read the length then one needs to convert the length into centimetre and

once this conversion is done then one needs to write the length in centimetre either on output in output file or on the screen in the display. So, how do we achieve this set of 7 steps using Fortran. So, that is what we are going to see in the next couple of slides.

(Refer Slide Time: 15:18)

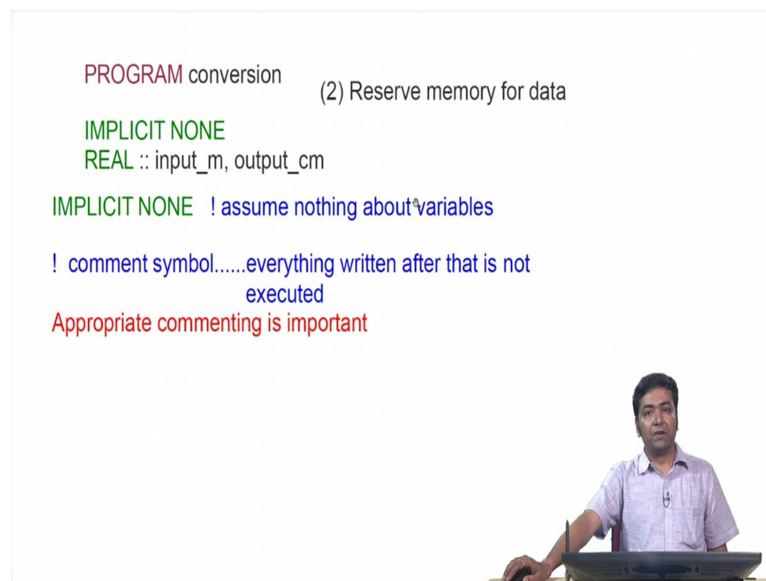
So this is what a typical program or Fortran program will look like which will perform the jobs which we have listed here, so, from 1 to 7. So, if we now go one line at a time of this Fortran file.

(Refer Slide Time: 15:35)



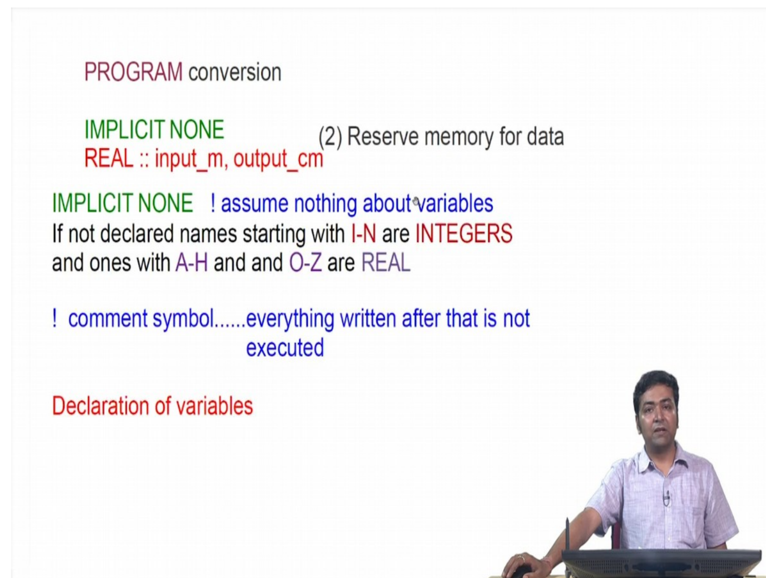
So, the first line which we have is the probe we have a Fortran keyword which is called the program this keyword tells the compiler that the program file starts from. This is the beginning of the program and then we have a name here followed the Fortran keyword program is followed by a name, typically one should choose a name which can tell anyone that what is the job of the of this present program. So, this is my first tip that is the start of the program.

(Refer Slide Time: 16:04)



Now, in the second step I as I mentioned we reserved memory for data and this is done in the following way. So, I have used two Fortran lines, the first one is called the 'implicit none' and second is this type of Fortran thing which is called 'real'. So, for implicit nine none what it tells the code is that I do not assume anything about the type of variables.

(Refer Slide Time: 16:39)

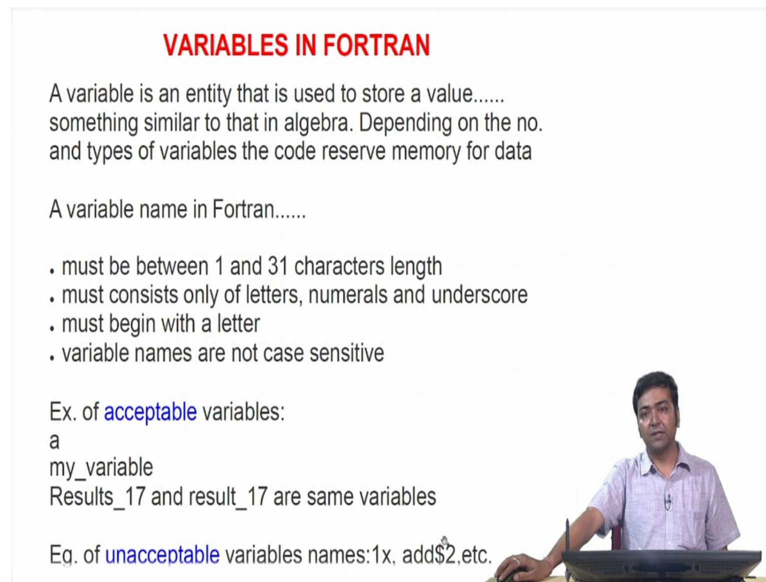


So, the reason you need to write implicit none is for the following, if you do not write implicit none then Fortran inherently assumes that all variables which are declared with alphabet starting from 1 to N are integers and the ones starting from A to H and O to Z are real numbers. So, to have a control over the variables, so, it is essential to tell the compiler that it does not intrinsically assume anything and then the second line which is this one 'real' which contains the 'real' word.

So, this is again a Fortran keyword and this particular step is called the declaration of variables. Now, for this present purpose I need two variables; one is to store the input number given by the user and secondly, to store the output number which the code generates and both these numbers can be real numbers. So, I have declared that two variables input underscore in input underscore meter and output underscore centimetre and I have declared them as real numbers here. So, also one should note that one should choose the variable names wisely, so, that they can give us an indication of what information is stored in that variable.

So, for example, if the moment I see that this variable name here input underscore m. So, this tells me that two crucial information's; one is this is this contents and input variable this is an input variable and it stores the number in meters. Similarly, for the output underscore centimetres it shows me that its a output variable and then the numbers is in centimetres.

(Refer Slide Time: 18:22)

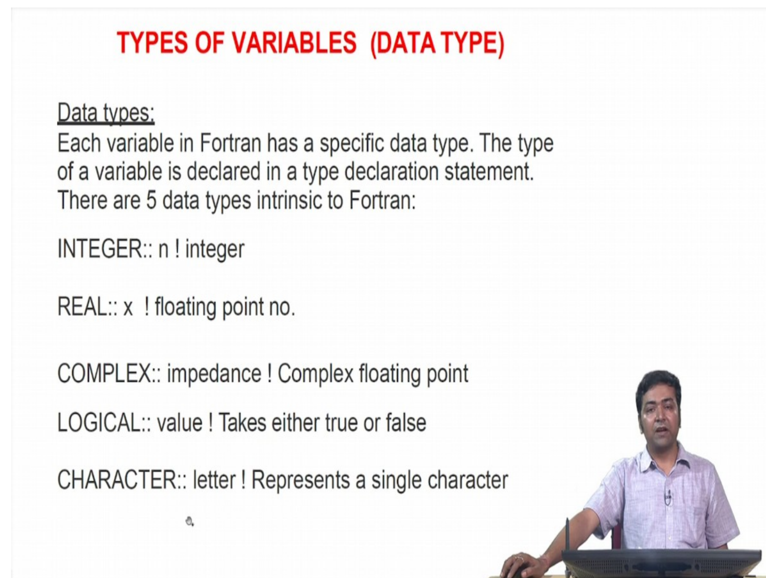


So, there are different types of variable declaration in Fortran, but a Fortran variable name has to satisfy these following criteria; one is usually the length of a Fortran variable is anywhere between 1 and 31 characters and the Fortran variable name consists only of letters, numerals and underscores.

Each Fortran variable name should must begin with a letter and the variable names are not case-sensitive. So, here are some examples of acceptable variable names. So, for example here 'a' then 'my underscore' variable, then 'results underscore' capital results starting with capital R underscore 17 and then result starting with small r underscore 17. So, Fortran will identify both of these to be the same variable.

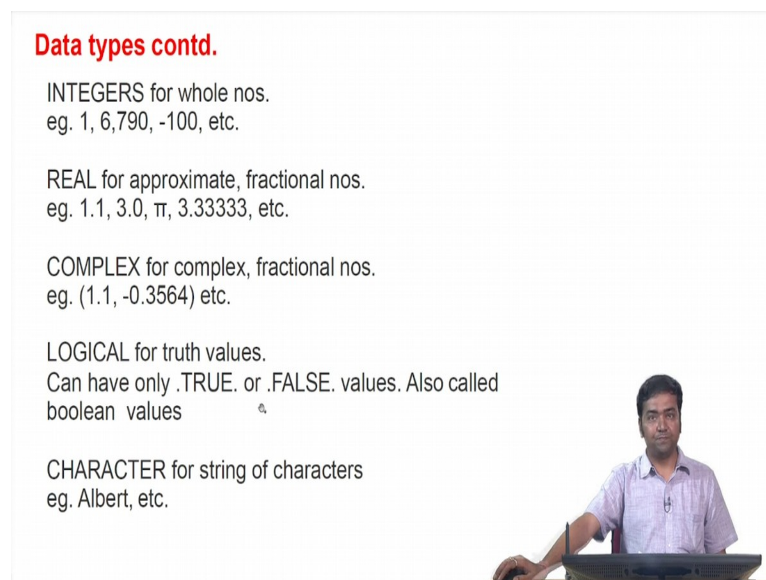
So, some examples of unacceptable variables here are for example, in this one where my variable name is starting with a number and then in the second one where I have these special characters inside by variable names and special symbols inside the variable name.

(Refer Slide Time: 19:38)



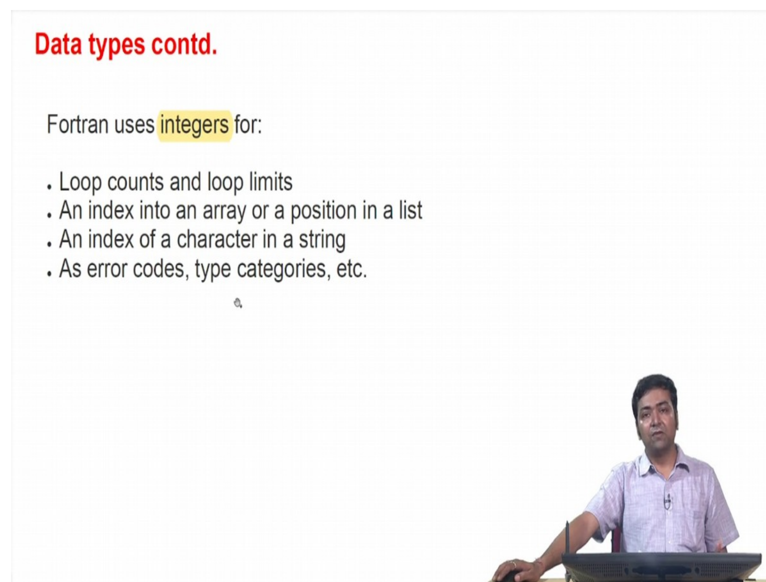
So, the variables can be classified typically into 5 classes which are called the data types. So, if a number is an integer it is classified by this integer keyword Fortran keyword, if it is a floating point number it is classified by the real keyword, if its a complex floating point number then its classified as with the complex keyword and then if it just takes logical value that is either whether a statement is true or false. Then it is classified it is indicated by a logical Fortran keyword and you can also store characters which is typically denoted by the character keyword.

(Refer Slide Time: 20:25)



So, here are some examples of what I just now said for example, integers which are whole numbers they are typically 1, 6, 790 and so on and so forth. Real numbers are either approximate numbers for or approximates for fractional numbers for example, 1.1, pi 3, 3.333 etcetera, complex is for complex fractional numbers for example, it has two parts. The first part is the real part and then the second part is the imaginary part, then we can have true and false value for logical and then for character for example, I have put this name Albert for which can.... so, basically this variable contains a string of characters A l b e r t.

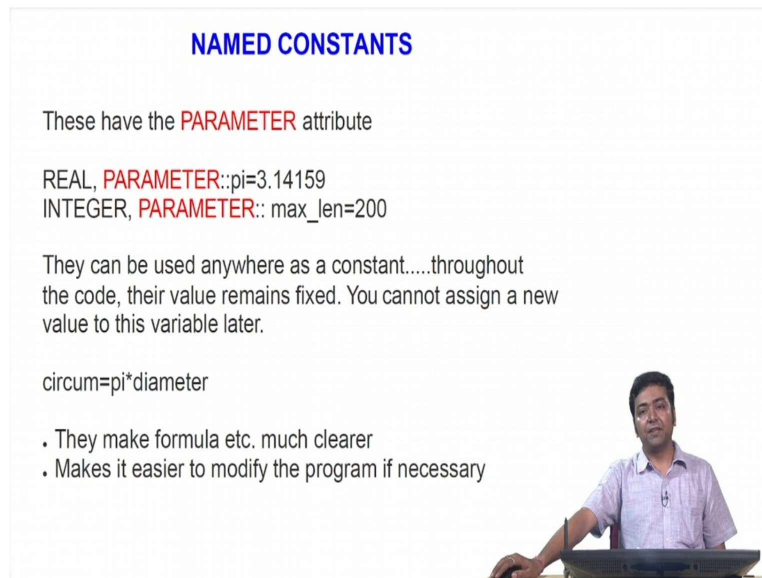
(Refer Slide Time: 21:10)



Now, when are typically Fortran integers used? Fortran integers are used typically for the following cases. So, for example, if I want to have a counter over my loop or so we will see what I mean a loop shortly or if I put an end to the loop.

So, then one typically uses Fortran integers, then if we want to refer to an index in an array or a position in a list or an index in a character or a string or for error codes and the different types of categories. This is these are the typical scenarios where one uses Fortran integers.

(Refer Slide Time: 21:50)

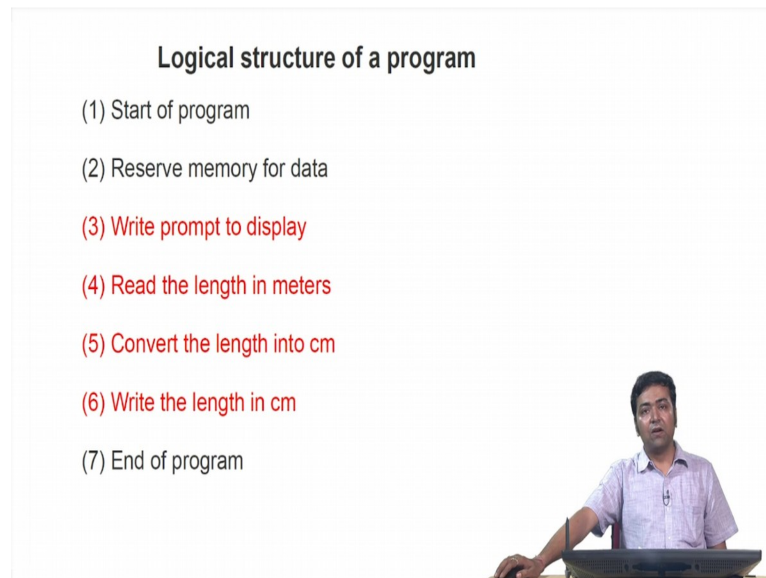


Now, many times what happens is we have to use same number repeatedly in different places of my code instead of setting up a variable for that and assigning the value to that variable repeatedly. So, what one can do is one can use this parameter attribute. So, this parameter attribute fall in the class of named constants. So, the way to use it is suppose I want to store the value of pi and I want to use it in n different places in my program.

So, the way I will do it is at the beginning of my program I will declare a variable which I called as pi and then I declare it as a real type of variable and then I use this attribute parameter along with my variable declaration and then I set the value of pi 2 the number of to the actual number. So, once this is done, so then wherever I use this variable pi in my code, the code will always take this value. Similarly you can do the same thing with other variable types also for example, integer.

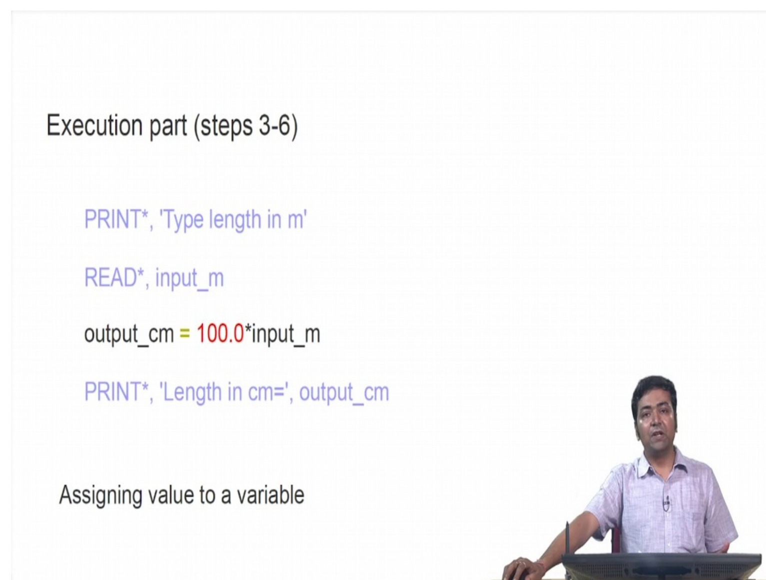
So, the basic idea is you can once you declare a variable name with a parameter attribute this can be used as a constant throughout the code and their value remains fixed you cannot assign a new value to this variable later. So, the advantage of using this as I also mentioned before is that they make the formula much clearer, it makes it easier to modify the program when later if it is necessary. So, you just change at one place that is at the beginning of the program and then everywhere things get changed.

(Refer Slide Time: 23:36)



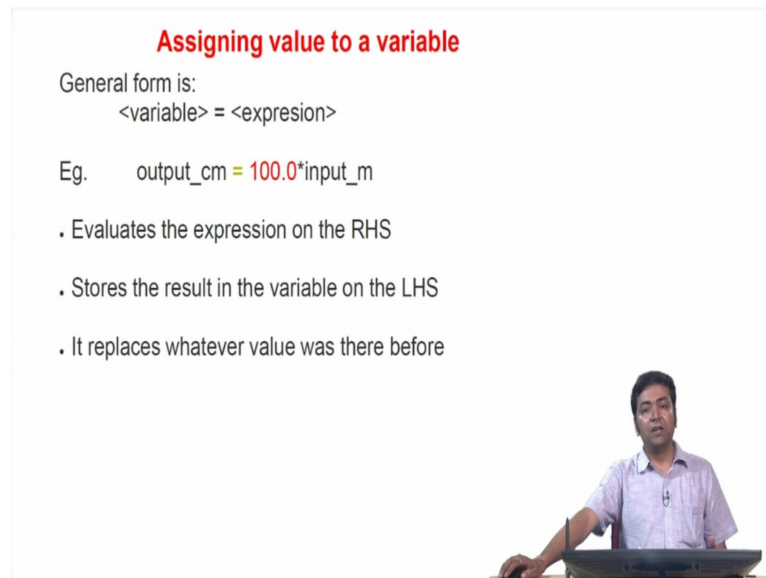
So, we have seen the first and the second step now we look in two steps 3 to 6 of my list. So, the next step is how do I write or prompt to a display?

(Refer Slide Time: 23:51)



So, typically, so, these set of instructions which I have written here consists of steps 3 to 6 of my logical structure of the program and so, what that implies is? So, now, what we are going to discuss is how to read inputs and how to write outputs and then how to perform mathematical operations inside the code. So, before we go into how to read and write outputs. So, first let us see how we assign a variable a value to the variable.

(Refer Slide Time: 24:27)



So, our general form is that on the left hand side you have a variable name and on the right hand side you have the expression which you are using to create that were to or to generate that value for the variable. So, for example, in this statement on the left hand right I have the variable name here to which I store the I want to store the product of 100 into my input value which is in meter.

So, these are typically the steps, so usually the code evaluates the expression on the right hand side stores the result on the left hand side and once its when it stores the result on the left hand side if initially there was some value assigned to this variable it overrides that.

(Refer Slide Time: 25:10)

Basic operations:

- = assignment
- + addition
- - minus or subtraction
- / division
- * multiplication
- ** exponentiation, i.e. raised to the power of

Exponents may be of any arithmetic type, INTEGER, REAL or COMPLEX

So, the basic mathematical operations one can perform using Fortran are typically the assignment which is done by the 'equal to' sign, then one can do addition, subtraction, division, multiplication, exponential and they can do this three types of mathematical operations that is integer based mathematical operations, real based or complex based or a mixture.

(Refer Slide Time: 25:31)

Operator Precedence

Fortran uses the normal mathematical conventions:

```
B   E   M  D  A  S
()  **  *  /  +  -
```

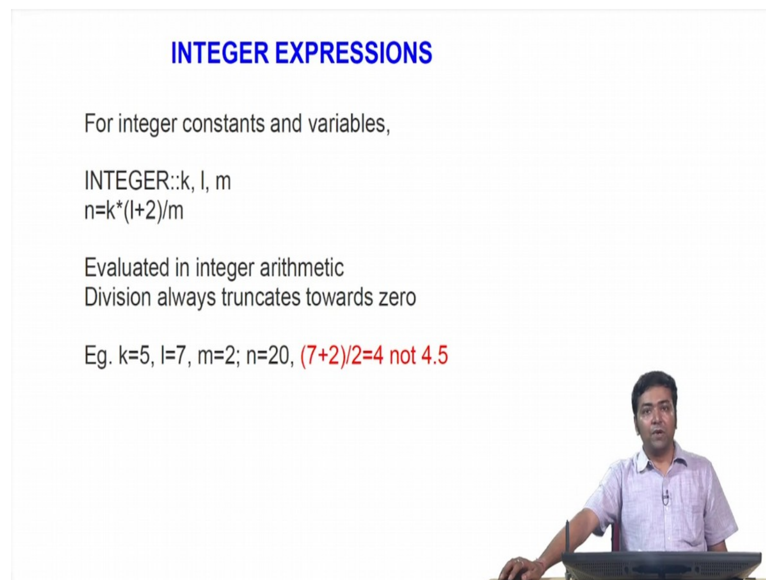
You can use parentheses to control the operation

Always use parentheses in case of ambiguity

So, the for the operators are typically used in the following precedence. So, first it will do the actions which are within the bracket, then followed by the exponential, then the product, then the division, then addition and then subtraction.

But it is always if you have in a statement in a single line if you have several of these operations together it is always advisable to use parentheses to control the not only the controlled operation, but also it will make clarity in your program and you will be able to remove the ambiguity in your program and as a result you can make you will make less mistakes.

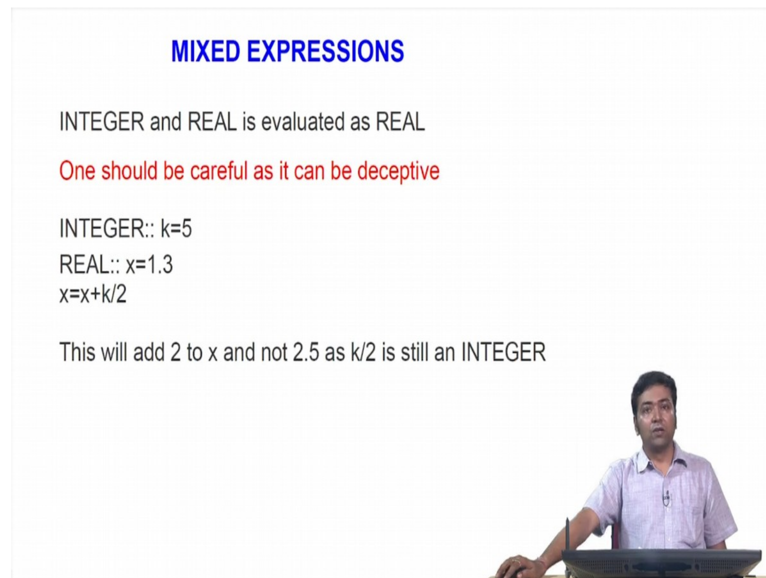
(Refer Slide Time: 26:14)



So, one thing one needs to be a bit careful is when one does these integers one one uses integer expressions. For example, if I have these three variables k l and m which I have declared as integer type and I want to perform this expression k into l plus m l plus 2 by m and which I want to assign to this variable n if we do a normal mathematics we will get we would expect.

So, if we take these values of say k equals to 5, l equals to m, 7 m equals to 2, n equals to 20. So, what we will expect is in a normal mathematics is a value of 4.5, but if you do the integer arithmetic you will get a value of 4. So, when one deals with mathematical expressions using integer variables. So, one should remember that Fortran always does the integer mathematics with that. So, one should be careful about this.

(Refer Slide Time: 27:16)

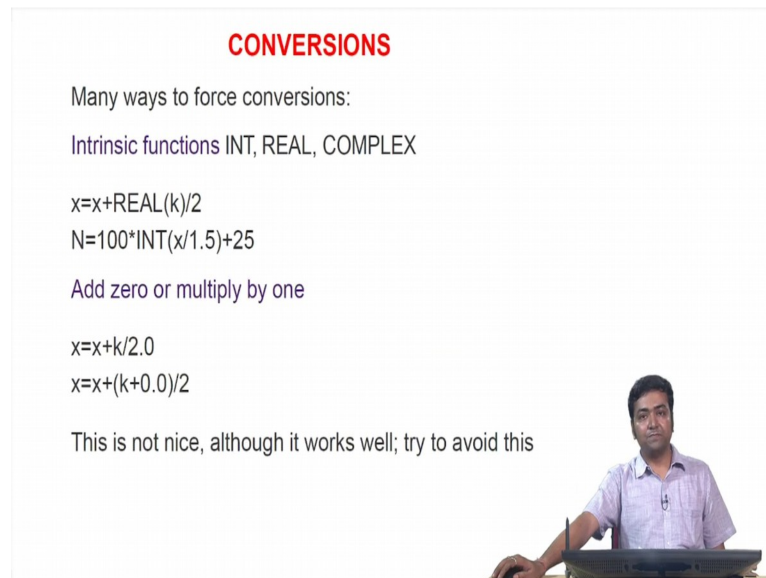


So, as I mentioned before also that Fortran allows integer and a mixing of integer and real numbers. So, it typically it should be avoided because of there is example which I will give below. So, for example, I have integer variable which is k equals to 5, then I and I have a real variable which is k equals 2 where I assign the value x equals to 1.3 and I want to perform reassign the value of x 2 x plus k by 2. Now one might think that if I do a simple maths my k by 2 will be 2.5 and then if I add 1.3 to it I will get 3.8.

So, I would expect k 2 x to be 3.8, but in reality since my k is an integer number and I am dividing it by 2 here which is also an integer number. So, what the actual value of this division will be 2 and not 2.5 as one would expect because this is a integer division and then this integer number the output of this integer division two which we get that is added to 2 to the real number 1.3

So, the message which I am trying to give here is that when you mix different types of variables a real variable integer variable and use them in mathematical expressions. So, one should be very careful about what one is doing otherwise one might result in chunk numbers.

(Refer Slide Time: 28:51)

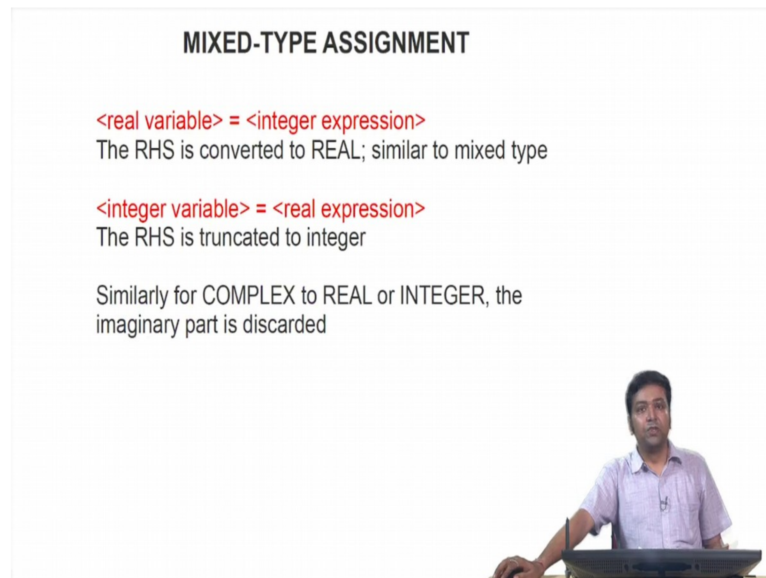


Another way to use this is we can mix we can convert integers to real and real to integers and vice versa. So, for example, if I have here I have a my variable which is k which is an integer number and I want to convert it to real number. So, I can do it in this way I as use this Fortran attribute real and then within bracket I write k.

So, now, if I do the division here, so, I will get 2.5 another way similarly to convert a real number to an integer one can use this Fortran statement called int. Another way to do it is you can multiply we can add a 0 or multiply by 1. For example, in the previous slide when I showed that k by 2, so here both are integer numbers.

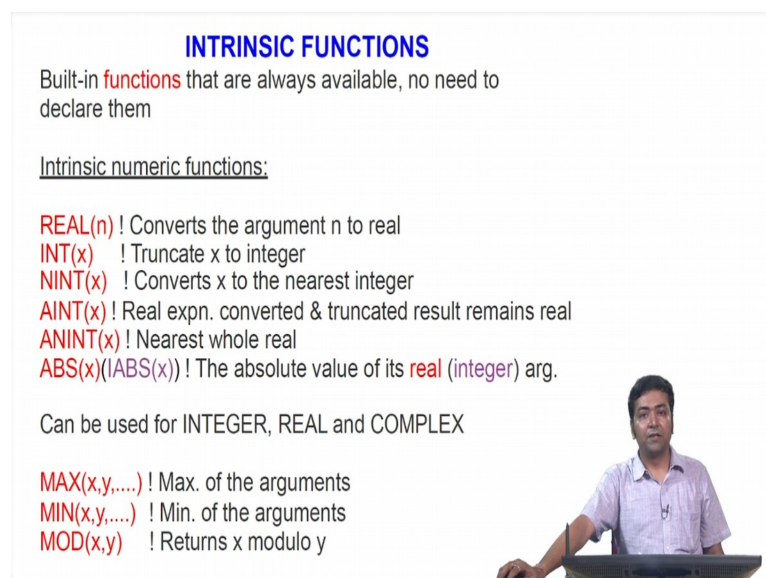
Now, instead of I write instead of k by 2, I write k by 2.0, then 2.0 is real number and when you do operation with consisting of a integer and a real number you do a real operation. So, this is another way of converting real number to an integer or an integer number to real, but this is not advisable because again you this makes you prone to mistake. So, the correct way to do it or the proper and safe way to do it is using this Fortran statements.

(Refer Slide Time: 30:17)



So, one can also do a mixed type of assignments for example, you can assign in integer expression to a real variable and the vice versa similarly between complex to real and integer. So, when you do complex to real or an integer assignment that is in my expression on the right hand side is a complex number and I want to assign it to a integer or a real variable. So, what it takes is just the real part of the imaginary number, the imaginary part is discarded.

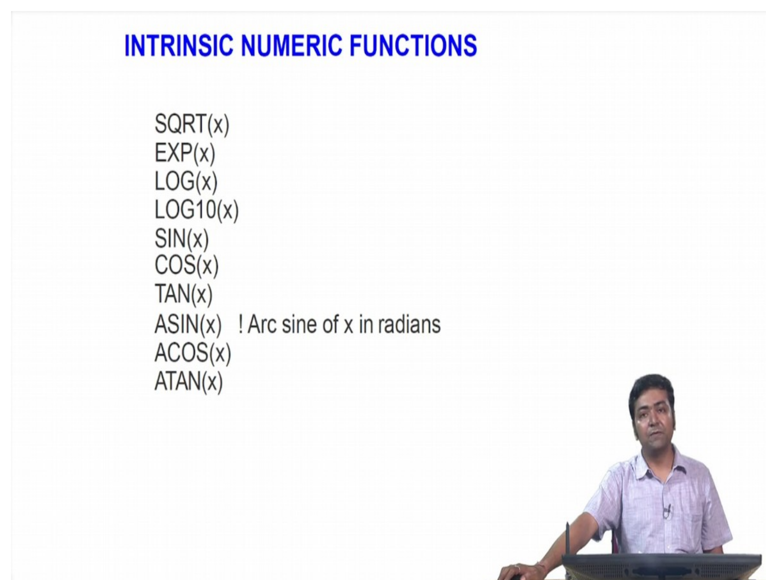
(Refer Slide Time: 30:48)



There are some basic intrinsic functions in the Fortran program which allow you to do these conversions which I have listed below. So, for example, the real function it converts the argument into a real number, INT converts its to a integer number NINT converts a real number to the nearest integer, AINT converts exponent real exponential and truncate the results to a real number and so on and so forth.

Also suppose if you have a set of numbers and you just want to find the maximum value or the minimum value or you want to do a module operation you can use these Fortran intrinsic functions. For example, max gives you the maximum value of a set of numbers given in the argument min gives you the minimum value and mod gives you the x modulo of y.

(Refer Slide Time: 31:37)



So apart from that there are also other intrinsic functions some geometrical functions or mathematical functions like square, root, exponential, logical, a logarithmic base, x logarithmic base, tan, sin function, sin inverse, cos inverse and so on and so forth. So, there is a whole lot of things mathematical functions which are available in Fortran which you can use there.

(Refer Slide Time: 31:57)

RELATIONAL OPERATORS

Relations create LOGICAL values

These can be used on any other built-in type
== (or .EQ.) equal to
/= (or .NE.) not equal to
These can be used only on INTEGER and REAL
< (or .LT.) less than
<= (or .LE.) less than or equal
> (or .GT.) greater than
>= (or .GE.) greater than or equal
.NOT. complement or negation
.AND. logical intersection
.OR. logical union

Apart from the mathematical functions you can also give the logical values or logical expressions. For example, so, you can use double equal to or you can also write as dot EQ dot to say that with this is equal to operation similarly dot any or a slash equal to say that this is not equal to operations.

And then you can also do less than; less than equal to greater than greater than equal to ah. So, all these type of logical operations are performed using these symbols you can also perform negation or logical intersections and logical union operations also by using the NOT and AND the OR functions in the Fortran port.