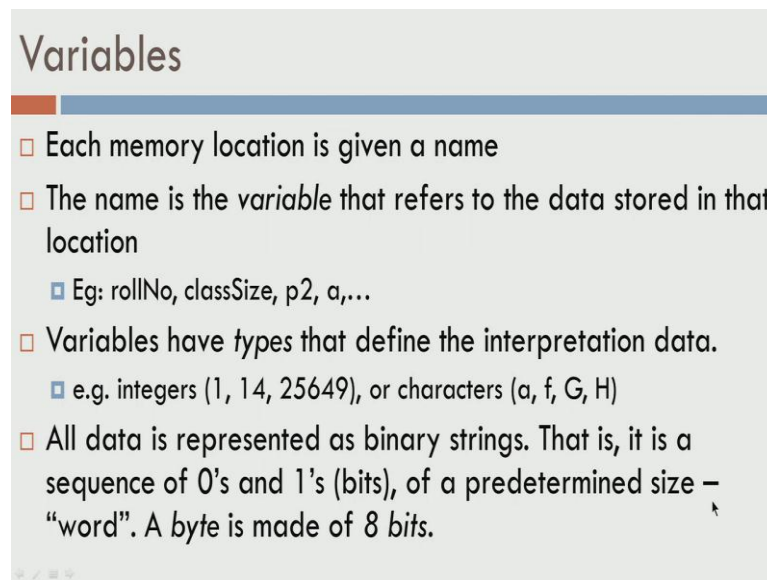


Programming, Data Structures and Algorithms
Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 04
How a Program Runs
Assignment operator
Variables and constants
Variable declarations and data types
Arithmetic operators in C; Precedence
Increment and decrement

We use this 7 variables a, b, c, d, p 2, p 1, p naught and these are called variables.

(Refer Slide Time: 00:12)



Variables

- Each memory location is given a name
- The name is the *variable* that refers to the data stored in that location
 - ▣ Eg: rollNo, classSize, p2, a,...
- Variables have *types* that define the interpretation data.
 - ▣ e.g. integers (1, 14, 25649), or characters (a, f, G, H)
- All data is represented as binary strings. That is, it is a sequence of 0's and 1's (bits), of a predetermined size – “word”. A byte is made of 8 bits.

So, every memory location is given a name. So, if you have a variable you use a name and this name is something that you gives so, that you can remember what it is about and in turn this variable name is attached to a memory location, when you run the program. So, the name is,. So, when you say a the variable name a, you are actually referring to the value at location that we are going to call a. And when you say a times b, it is actually going to take the value at location a and value at location b and multiply them together.

So, the name is actually referring to the data and you usually have names which make some sense to you. So, it could be roll number, class size or in our example we use p 2, a, b etcetera. So, these variables have some kind of data type attached with them. So, in our example we expected a, b, c, d to be integers or we wrote the program to be in such a way that they are integers and p 2, p 1, p naught are also integers.

So, because they are integers they cannot have something like a string, let us say it can not have my name assigned to a, a equal to Shankar does not make sense, a should be a number. Then, all the data internal to the computer is actually represented as a sequence of 1s and 0s of some particular size or some predetermine size, which we will call word. So, this integer is supposed to be of certain number of bytes and usually integers of size 4 bytes.

(Refer Slide Time: 02:04)

Instructions

- Instructions take data stored in variables as arguments.
- Some instructions do some operation on the data and store it back in some variable.
- The instruction " $X \leftarrow X+1$ " on integer type says: "Take the integer stored in X , add 1 to it, and store it back in (location) X ".
- Other instructions tell the processor to do something. For example, "jump" to a particular instruction next, or to exit

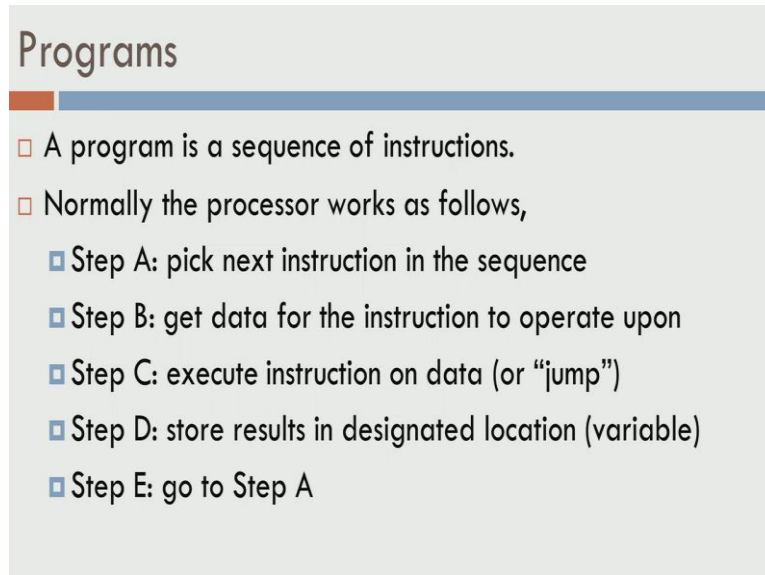
Then, there are these various operations that we did. So, if you look at a times c which is assigned to p 2, then the star is an operation, the assignment itself is an operation and so, on. So, instructions take the data that is stored in the variables as arguments and some of these instructions actually perform operations. For example, if I do x equals x plus 1, it takes the integer x add 1 to it and the result is then put back in the same location x.

And some other instructions actually do change the order in which the program is run and these are called control instructions, we will see them in a little more detail later. So, I want you do look at this x equals x plus 1 in little more carefully. So, if you do basic algebra you may be confused about this statement. So, let us say x equals x plus 1 is written in a C program and you look at it, if you do basic algebra from a 6th grade or 7th grade, you would cancel x on both the sides and you would be looking at 0 equals 1.

So, but that is indeed not the case. So, the assignment operation is something which is different, do not read the sign equals to as though it is an equality. So, the way it should be looked at is, it is a statement it has to executed and the way it is done is, you execute

whatever is on the right side, you get a result from there, you take that result and put that result on the variable on the left side.

(Refer Slide Time: 03:41)



Programs

- A program is a sequence of instructions.
- Normally the processor works as follows,
 - ▣ Step A: pick next instruction in the sequence
 - ▣ Step B: get data for the instruction to operate upon
 - ▣ Step C: execute instruction on data (or “jump”)
 - ▣ Step D: store results in designated location (variable)
 - ▣ Step E: go to Step A

So, let us look at the program, a program is a sequence of instructions. Normally, the processor will work in the following sequence of steps. Step A would be pick the next instruction in the sequence, then step B would be get data that is required for the instruction to operate upon, execute the instruction on the data. So, if it is star then you do multiply, if it is plus you add and so, on. And a step D would be take it and store it back as a result somewhere, it could be either a memory location or it could be some internal storage, like 4 and 6 were stored in our steps and finally, go back to step A itself look for the next instruction and so, on.

So, this is going to be repeated in a loop. So, you pick the instruction, you pick the data that is required for it, you do the operation, store the result, go back and pick the next instruction and keep doing this till there are no more instructions for the particular program.

(Refer Slide Time: 04:42)

Assignments

- = is the assignment operator
- The value of a variable is modified due to an assignment
- LHS has the variable to be modified
 - RHS is the value to be assigned.
- RHS is evaluated first
 - After completing the operation on RHS, assignment is performed.
- $a = 1$
- $a = c$
- $a = \text{MAX_PILLAR_RADIUS}$
- $a = a*b + d/e$

So, let us look take a careful look at what the assignment itself means. So, this equality or the equal symbol that you saw is called the assignment operator. So, when you see $p2$ equals a into c , you multiply a with c and the result is assigned to the variable on the left side called $p2$. Therefore, we call this equality an assignment operator, it takes the value and puts it in the memory location $p2$. Therefore, it is doing an operation which is a memory operation, in this case it is a memory write operation, you are writing to memory. So, it is an assignment operator.

So, the value of a variable could be modified due to an assignment. So, the way it works is the left hand side is the variable to be modified and right hand side is the value to be assigned. So, you have variable name equals value. So, if you have a equals to 1 as the assignments statement, then 1 is the value that get's assigned to a . If you have a equals to c , c could itself be another variable, you do not copy the character c into a , instead you go and look at the variable called c , look at the value that c contains and copy that value into a .

If you have something like a equals a into b plus d by e , then you actually take the variable value contained in the variable a , multiply that by the value contained in the variable b , store it temporarily. Then, take d and e the values contain in these variables divide d by e , store it again temporarily at these two and put it back the memory location a itself. So, the process is the right hand side is evaluated first. So, you would evaluate this right hand side first and after completing and only after completing all the operation on the right hand side, the assignment operation is perform this will write the result back

on to the left hand side.

(Refer Slide Time: 06:42)

A presentation slide titled "Variables and Constants" with a sub-heading "Names". The slide lists several rules for naming variables: they must be made up of letters, digits, and underscores; they are case sensitive (e.g., classSize and classsize are different); they have a maximum size of 31 letters; the first character must be a letter; and they should be meaningful and self-documenting. Examples are given: "PI" is a constant and "radius" is a variable. It also notes that keywords like "if", "for", "else", and "float" are reserved. The slide number "36" is visible in the top left corner.

So, variables and constants are two typical things. So, this is where I want to bring this distinction between, what is a variable, and what is a constant. So, a variable is something that can change during the execution of a program, whereas a constant can not change its value during the execution. So, you look at `a = 1`, `a` is a variable here, `1` is number 1, number 1 is not going to be changed during the execution of the program. Therefore, `1` is a constant and `a` is a variable.

So, the variable names are made up of letters, digits and underscore. So, these are called identifiers. So, for different memory locations you give different identifying names, just like we have names, variables are the names for the memory locations. And the variable names are what are called case sensitive, what that means, is upper case or capital letters are different from lower case or small case letters.

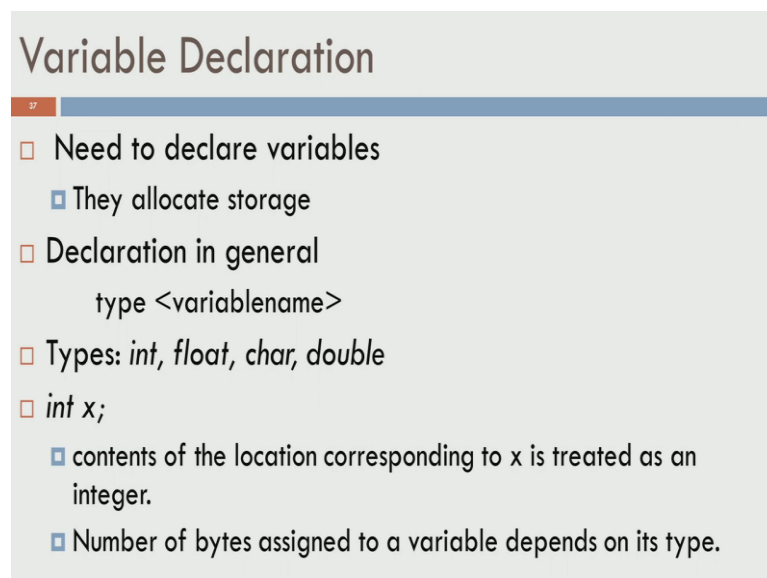
So, if I use the variable name called class size with a capital S, it is going to be different from a variable called class size with a lower case s. So, these two are two different variables, which means they will get two different memory locations assigned to them. And the maximum size, that you can have for a variable name is 31 letters, you cannot have more than 31 letters for a variable. The first character for a variable that you have, that you write must be a letter, it cannot be underscore or it cannot be a digit it has to be a letter.

Typically, you use a meaningful and self-documenting name for the variables. So, for

instance let us say I want to use pi or pi. So, pi is a constant, pi is not going to change its value, it is going to be 3.141 and. So, on, it is a constant. So, usually constants are given names which are all capital letters, whereas variables are given names which are usually a mix of lower case and upper case letters.

And then there or key words are words that are reserved by the programming language, we already saw if statement in module 2, there are also other statements, other key words like for and so, on, we will see them later. So, these are not names that you can assign to your variables, you cannot have a variable by name if or for or else or float or int or while and so, on.

(Refer Slide Time: 09:15)



Variable Declaration

- Need to declare variables
 - ▣ They allocate storage
- Declaration in general
 - type <variablename>
- Types: *int, float, char, double*
- *int x;*
 - ▣ contents of the location corresponding to x is treated as an integer.
 - ▣ Number of bytes assigned to a variable depends on its type.

So, the variable declaration goes as follows, once you declare you get storage already showed that in the animation. Declaration is the general form, you have the data type followed by the variable name or a list of variable names. The various types that are allowed are integer, float, character and double, int is for integers. So, you can only store integral values in them, float and double are for real values and you can store a dot something is a 5.34, 3.14 and so, on, char is for character it can store a single letter, single letter like a, b, c, d or such things.

So, if you look at the statement `int x;` what it really does is you have a variable by name x and it is of data type integer. And there is a memory location, which is labeled x for this execution of the program, it assigns a creation number of bytes for this. Whenever, you use the variable x in your program, it will during the execution this value,

which is contained in this location x is used.

(Refer Slide Time: 10:28)

Modifying Variables

- Each C program is a sequence of modification of variable values
- A modification can happen due to operations like $+$, $-$, $/$, $*$, etc.
- Also due to some functions/operators provided by the system like `sizeof`, `sin` etc.
- Also due to some functions (another part of the program) created by the programmer.

So, we actually use the term variable, because the value of a variable can change during the execution of the program. So, a program is essentially a modification of variable values. So, for instance even the variables a , b , c and d they had some unknown values initially and they changed only when the user input those values. Similarly, p , 2 , p , 1 and p naught, were unknown values initially and they changed once the operations on the respective right hand side where over.

So, each C program is a modification of variable values and the modification can happen due to operations like, plus, minus, slash which is for division, star for multiplication and so, on. These variables can also change values, because of functions or operators provided by the system. So, for example, I could say a equals \sin of x . So, x would be treated as a parameter to the function called sine, sine is a built in function in the math library, it will do sinusoid or sine of x . The result of sine of x will be stored in this variable called a or this variables can change due to some functions itself, which the programmers create. We have not written any such a functions yet, we will see them as we go along.

(Refer Slide Time: 11:53)

Operators in C

Four basic operators

$+$, $-$, $*$, $/$

addition, subtraction, multiplication and division
applicable to integers and floating point numbers

integer division - fractional part of result **truncated**

$12/5$ is 2, $5/9$ is 0

modulus operator : $\%$

$x \% y$: gives the remainder after x is divided by y
applicable only for integers, not to float/double

Let us look at the kind of operators in C, there are four basic operators for arithmetic namely, plus, minus, star and slash, they stand for addition, subtraction, multiplication and division respectively. You can use these operations against integers and floating point numbers. So, plus and minus for integer and floating point will just add the values or subtract the values, star and slash have a little bit of meaning or a change in meaning when you attach them to integers as opposed to floating point.

When we do integer division, the fractional part of the result is truncated if you do integer division. So, for example, 1, 2 or number 12 is an integer and number 5 is an integer. So, if you write 12 slash 5, you are dividing 12 by 5 and even though the result is 2.4, this 0.4 is truncated. So, the result of dividing an integer by another integer, this is also an integer in this case it is only 2.

So, a more drastic case is, if you divide 5 by 9 for example.. So, 5 is smaller than 9, so, when you divide 5 by 9 you have a fraction and know integer value. So, the result would be 0. So, you have to watch out for this, when you program, but for now this is not a problem and then there is this modulo operator or percentage. So, this percentage if you write x percentage y on the right hand side of assignment statement, what it will do is, it will take x and divide it by y .

But, instead of putting the quotient it finds out the remainder. So, a by b finds out the quotient and a percentage b finds out the remainder. So, this is for integers. So, of course, for floating point numbers a percentage b does not make sense, you can take any floating

point number, divide by another floating point number and you can always get a coefficient with the modulo being 0. So, this is called the modulo operator, because it finds out the remainder. So, percentage is defined only for integers, it is not defined for floating point.

(Refer Slide Time: 14:12)

Operator Precedence

first	parenthesized subexpressions - innermost first
second	*, / and % (associates left to right)
third	+ and - (associates left to right)

$$a + b * c * d \% e - f / g$$

4 1 2 3 6 5

$$(a + (((b * c) * d) \% e)) - (f / g)$$

good practice -- use parentheses rather than rely on precedence rules

Let us look at these operators, if you look at operations just like in when you write expressions in your basic algebra, there are rules that you have to follow the rules of precedence. So, even in our example we had a times d plus b times c which was return as a into d plus b into c. So, what is the order in which things are done. So, the first precedence is for parenthesized expressions, the next precedence is for star and slash and modulo and the last level of precedence is for plus and minus.

So, to illustrate what I mean by that, let us take this expression, this complicated expression a plus b times c plus d and c times d and so, on. So, if you are not careful you may end up doing something like this. So, let say I start looking at it from the left side, I see a plus b I take a and b first add it and then I multiply the result by c I take the result multiply by d take that result find the modulo by e and so, on. But this is not the way in which we do this in algebra either.

So, in algebra if an expression like this is return, then you do not do the addition first, you go and look for higher precedence operations first followed by lower precedence operations. So, the numbers that you see on the right side or not the values attached to these variables, instead that indicates the order in which the operations will be done. So,

if this expression is given to you, the very first operation that will be done would be b times c. So, this star is the very first operation that will be done, the result of b time c will be store temporarily somewhere, that result will be multiplied by d. So, this star is the second operation that will be done. So, now, you have b times c times d the third operation that will be done is percentage. So, b, c, d percentage e will find the remainder of dividing b, c, d by e. Once that is done, then the next operation that is done is plus.

So, you will do a plus that whatever is in this expression here then finally, you do this division followed by the subtraction. So, this is the order in which things will be done. So, if you are looking for a parentheses expression which is equivalent to this, this should be the parentheses expression. So, you start with b times c the inner most thing is the evaluated first, then you have that times d, then you have that modulo e.

So, you will have a result from here that will be added to a and you do f by g and this whole expression will be... So, this f by g will be subtracted from this whole expression on this side. So, it is missing one parentheses here, so, we expect a parentheses to be here. So, this is the order in which thinks are done. So, use parentheses all way if you are in doubt, until you get familiar with the order of precedence. So, star slash and percentage has higher precedence over plus and minus and parentheses has higher precedence over star slash and plus.

(Refer Slide Time: 17:56)

Precedence – Another Example

$$\text{Value} = a * (b+c) \% 5 + x / (3 + p) - r - i$$

Evaluation order:

1. $(b+c)$ and $(3+p)$: due to brackets
2. $*$ and $\%$ and $/$ have same precedence: $a(b+c)$ is evaluated first, then $\text{mod } 5$. Also, $x/(3+p)$.
3. Then, the additions and subtractions are done from the left to right.
4. Finally, the assignment of the RHS to LHS is done.
5. $=$ is the operator that violates the left to right rule

So, let us see another example here,. So, a times b plus c percentage 5 plus x by 3 plus p minus r minus i, it look complicated enough, it is also not a very good thing to write. So,

I am giving it only for illustration purposes, you should not be writing such things in your programs. So, the evaluation order is b plus c and 3 plus p would be done first, because of the brackets that you have around them. And once you have it, star slash and percentage have the same precedence and you have star and percentage here.

So, star comes before percentage in the program order, if you look at it from left to right star comes before percentage. So, star will be performed before the percentage. So, a times b plus c is evaluated and then you do it modulo 5 over it. Similarly, when you have x by 3 plus p, 3 plus p is evaluated already is you will do x by that, at the end of it you will have some value for this whole expression, some value for this whole expression minus r minus i.

So, now you have only pluses and minuses you simply go from the left to right side and you will do this plus that, the result is then, you remove r from it and then you remove i from it. So, that will be the order in which you do things. And finally, the whole expression is evaluated and you have a value at the right hand side, that value is assigned to this variable called value which is on the left hand side.

So, equal to is the only operator that violates the left to right rule. So, star you do the calculate term something on the left side before you calculate something on the right side, percentage also you evaluate something on the left side before evaluate something on the right side. But, for the equal to operator you need to evaluate the right hand side get it completely evaluated and then the result is assigned to the left hand side.

(Refer Slide Time: 20:02)

Increment and Decrement Operators

- unusual operators
 - prefix or postfix
 - only to variables
 - can only be in the RHS of =
- ++ adds 1 to its operand
- -- subtracts 1 from its operand
- n++ increments n after its use
- ++n increments n before its use
- n = 4 ; x = n++; y = ++n;
 - After execution, x would be 4 , y would be 6 and n would be 6

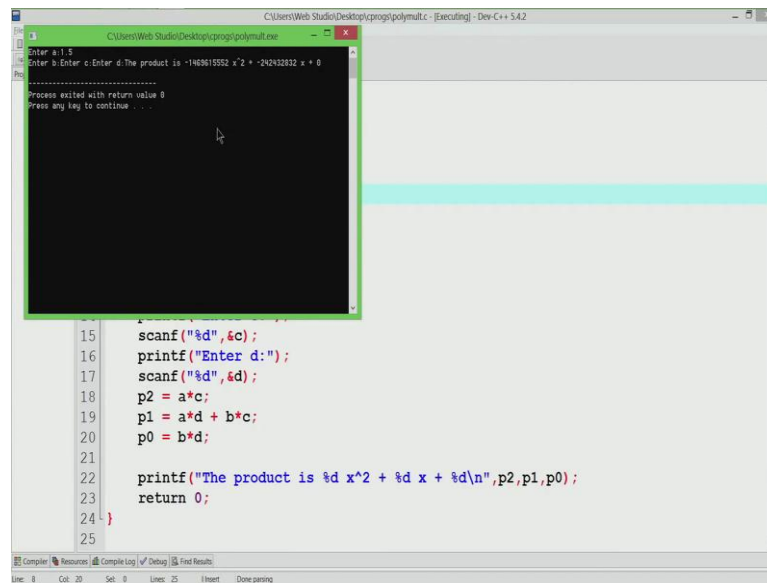
Let us look at a few other operators that are useful and very common in C and increment and decrement operators, these operators which are actually unusual not uncommon unusual. So, for example, this plus plus, see if you do a plus b it is addition of a and b, but if you do a plus plus it is actually equivalent to a equals a plus 1. So, plus plus means add 1 to the operand and minus minus means subtract 1 from the operand.

So, let us look at this small example n plus plus would mean, you take n add one to it and put the result back in n itself. So, it will increment n after the use of n, whereas plus plus n means increment before the use. So, let us see how this can be useful in an expression, how it can be used in an expression n equals 4. So, there is a right hand side which is 4, the left hand side as n, there is nothing to do on the right hand side it is a constant. So, 4 is assigned to n. So, there is nothing to do there, then let us look at this statement x equals n plus plus.

So, what we have here is this plus plus is a post increment operation what; that means, is use the current value of n, assign to the left side and then comeback and change the value of n. So, in this case if n equal to 4 was already executed, the current value of n is 4 it is assigned to x and then n is incremented to 5. So, at the end of these two statements n would be 5 and x would be 4. Then, if you have this third statement y equals plus plus n, n has a pre increment operator, which means n should be incremented before the use in the right hand side.

So, the current value of n is 5, you pre increment the result is 6 and that result is given to y. So, after the execution x would be 4, because you use the value before you incremented n, y would be 6, because, 2 increments happen before you assigned it to y and n would be 6, because n got to increments one as a post increment here and one has a pre increment here. So, remember the memory layout there are three variables n x and y which means there are three memory locations n x and y. And mentally simulate how this read and write on the memory is happening, involving the ALU. So, just remember that whenever you have a post increment, read assign to the left side and then do this operation. Whenever you have a pre increment, do the operation first and then assign to the left hand side so, it is as simple as that. So, we looked at the notion of variables and we looked that the notion of what the memory layout is, we also looked at what the various operators in C are the basic arithmetic operators and we looked at a small program.

(Refer Slide Time: 23:25)



```
15 scanf ("%d", &c);
16 printf ("Enter d:");
17 scanf ("%d", &d);
18 p2 = a*c;
19 p1 = a*d + b*c;
20 p0 = b*d;
21
22 printf ("The product is %d x^2 + %d x + %d\n", p2, p1, p0);
23 return 0;
24 }
25
```

enter a: 1.5
enter b: Enter c: Enter d: The product is -1403615552 x^2 + -242432332 x + 0
Process exited with return value 0
Press any key to continue . . .

So, I want to show something small about the code before I wrap up this session lecture 1. So, I am going to run this program, we already had this program, I am going to run this program, we did int a, b, c, d int p 2, p 1, p naught I am going to run this programs. So, let us what I am going to do is, I am going to run it with an unexpected input. So, I expect integers a, b, c and d to be given integral values, what if I violate that.

So, what I am going to do is, I am going to take two polynomials $1.5x + 1$ and multiply it with $1.5x + 1$ itself. So, I will try 1.5 and right away I have some result here with says the product is something, it did not even give me a chance to enter b, c, d and so, on, it gave me some product and clearly this is not what we expect. So, the variables a, b, c, d were integers and by just giving the input 1.5 somehow it got values for b, c and d incorrect values of course, and it gave me some polynomial here as a result also.

So, something bad happened here. So, will sit down and reason about this at a later point of time. So, this is just a show you that in module 2, we looked at some values that I gave to the program and for a, b and c it still gave me correct results. But, it is not because that it got the correct values in place, something else happened there will sit down and reason about that at a later point of time.

But, this is a program where entering a wrong data type 1.5 is a floating point, I gave it when an integer was expected and the program so, called crashed, it gave me something which is unexpected. So, with this we are at the end of lecture 1. So, see you in lecture 2.