Introduction to Problem Solving and Programming

Prof. Deepak Gupta

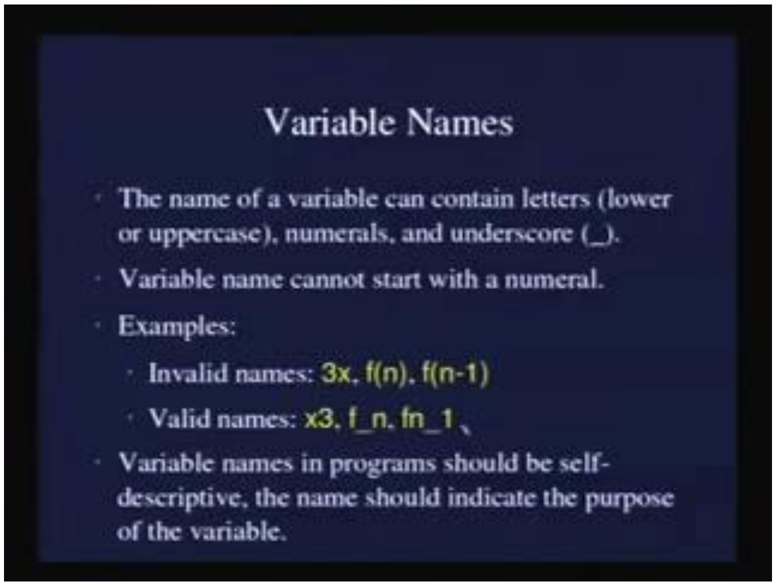
Department of Computer Science Engineering

Indian Institute of Technology, Kanpur

Lecture – 5

Hello, in the last lecture, we had looked at some C program and without really discussing the (()) formally, try to write some simple C programs and examine, how they works. And as I mentioned in the last class, starting today we will start discussing the C syntax and various integration of the language. From today, and today we will first look at variables and types.

(Refer Slide Time: 00:44)



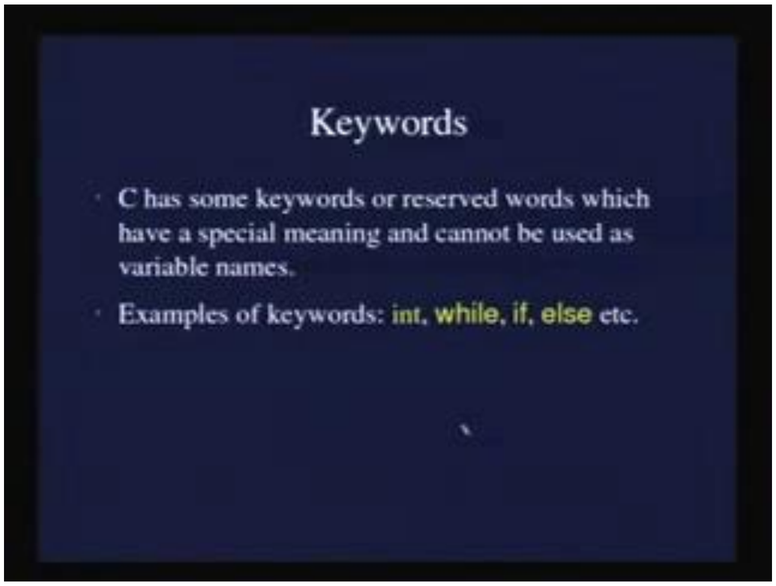
Variable Names

- The name of a variable can contain letters (lower or uppercase), numerals, and underscore (_).
- Variable name cannot start with a numeral.
- Examples:
  - Invalid names: 3x, f(n), f(n-1)
  - Valid names: x3, f_n, fn_1
- Variable names in programs should be self-descriptive, the name should indicate the purpose of the variable.

So we have already used several variables in the programs that we have written so far, but there are restriction from more the variable names can be, so far we have been using simple names like a, b, c, i and j. But actually the variables names can be pretty much as long as like but the restriction on the variable name is that it can only contain letter – lower or uppercase, that is a to z, or numeric – 0 to 9, or the special underscore character, it cannot contains any other character on the keyboard. In the addition, the variable name cannot start with a numeral that is it cannot start with a digit from 0 to nine.

So let’s see some examples of valid and invalid variable names. Here are some invalid variable names three x, f bracket n, f bracket n minus one, they are all invalid (( )) variable names. (( )) because it start with numeral number three; f n is illegal, because of this parenthesis character, the bracket character; this one is also illegal, because it has bracket character as well as minus sign. Note that the minus is the different from the underscore character. Here are some valid variable names. Instead of three x, you could have alter x three, here is an example where the variable name contains an underscore character – f underscore n or f n underscore one. So this is the restriction in post for the language and what are the language can be, but also when we are writing programs has, earlier also we must make sure that programs are readable, so that they can be later understood by ourselves and the other people who are reading the programs. And one very important aspect of program readability is that we should use descriptive variable names that is the variable names themselves show suggest what is the purpose of the variable.

(Refer Slide Time: 03:00)

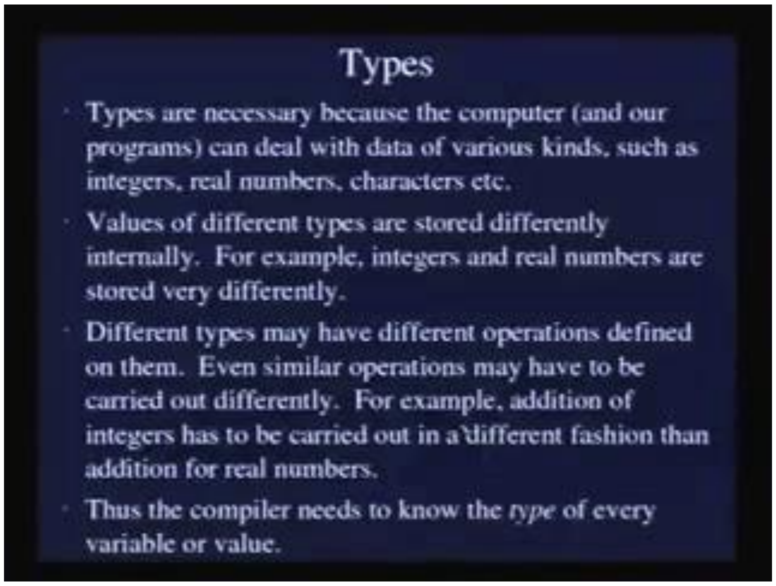


Keywords

- C has some keywords or reserved words which have a special meaning and cannot be used as variable names.
- Examples of keywords: int, while, if, else etc.

Here are some more constant on what the variables name cannot be, C has certain what are known as keywords or reserved words which cannot be used as variable names, because these are special meanings. We are already seen example of these keywords, they have appeared in or programs. Int, if you recall, it is a type integer; and so that is the keyword cannot be used as variable names. What is that means we cannot have a variable call int. And similarly, we saw that while, if, else etcetera, where special words has in our programs. So they cannot be used as variable name. But any other there are the whole lot of other keywords as well. But other than the keywords and the combination of characters containing only alphabets, numerals and underscore and not starting with numerals could be a valid variable name. But in the case again we should choose the variable name carefully, so that it represent the purpose of having that variable.

(Refer Slide Time: 04:02)



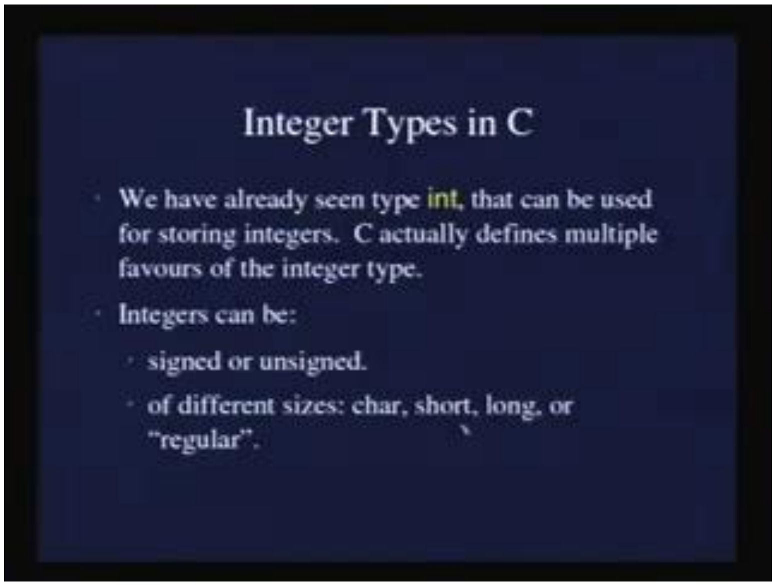
Types

- Types are necessary because the computer (and our programs) can deal with data of various kinds, such as integers, real numbers, characters etc.
- Values of different types are stored differently internally. For example, integers and real numbers are stored very differently.
- Different types may have different operations defined on them. Even similar operations may have to be carried out differently. For example, addition of integers has to be carried out in a different fashion than addition for real numbers.
- Thus the compiler needs to know the *type* of every variable or value.

Now let's look at types. We have already encountered types in the program that we have seen so far. We have already seen one types so far, which was integer types represented by the keywords int. But there are other types also available in C, large number of terms actually. So the types are necessary because the programs as well as the computer can deal with data of various kinds, integers are just one example, it could deal with real numbers, it could deal with strings made up of character and so on and so forth. And in general, the values of different types are stored differently on the computer internally. For example, integers and real numbers are stored very differently. We have seen already in earlier lecture that integers are stored in binary representation. We will shortly see roughly how the real numbers are stored in computer. And you will see that the way they are stored is very different.

And you can imagine different types may have different operations defined on them. For example, for integers regular Fourier numbers , we have the arithmetic operations of like addition, subtraction and so on. But for some other types, other operations maybe define which may not be extend for integers in general. Even similar operations in different types may be implemented differently internally in the computer. For example, addition define for both real numbers and integer, but because of the representation of integers and real numbers is very different. The computer has performed addition of integers very differently from the way perform addition of two real numbers. And therefore, the complier needs to know the type of every variable or value, that is being manipulated by the program, that is because you have to generate machine code to manipulate these values or the variables.

(Refer Slide Time: 06:11)

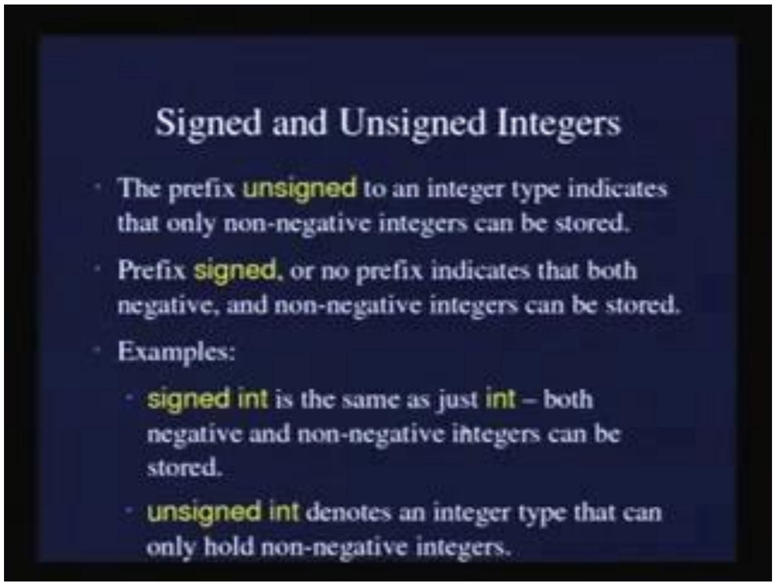


Integer Types in C

- We have already seen type int, that can be used for storing integers. C actually defines multiple favours of the integer type.
- Integers can be:
  - signed or unsigned.
  - of different sizes: char, short, long, or "regular".

So let’s first discuss integers types in C more details. We have already the type int, which can be used for storing integers, but as will shortly find out. C actually define number of favors in the integer types that is slightly different properties. So integers can be either unsigned or signed. What that means is that whether negative integers can be stored or not. When unsigned integers variable cannot stored in a negative integer value; whereas a singed integer variable can store both positive as well as negative integer value. Integers can be off different sizes, these are the various size of the integers in C, char, short, long or the regular integers. We will soon see what is different sizes in C, and what is the technical application of this.

(Refer Slide Time: 06:58)

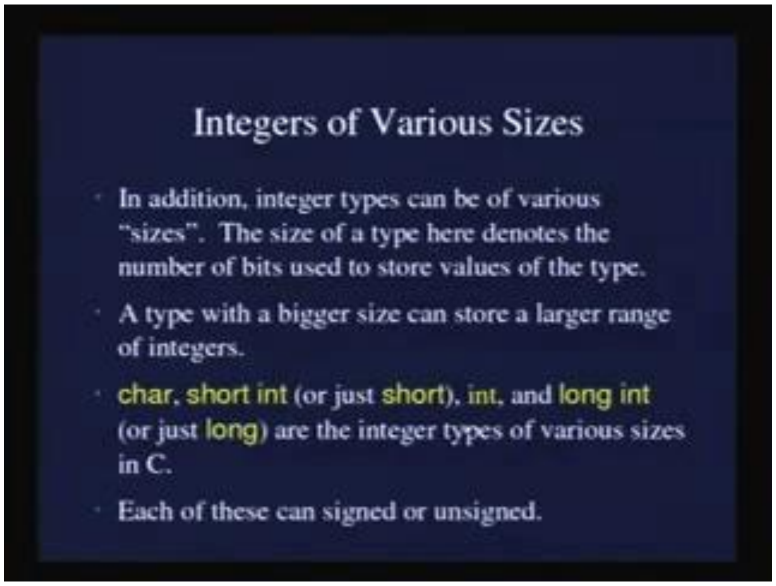


Signed and Unsigned Integers

- The prefix unsigned to an integer type indicates that only non-negative integers can be stored.
- Prefix signed, or no prefix indicates that both negative, and non-negative integers can be stored.
- Examples:
  - signed int is the same as just int – both negative and non-negative integers can be stored.
  - unsigned int denotes an integer type that can only hold non-negative integers.

So let's first focus our intention on signed and unsigned integers. So if the prefix the word was unsigned, unsigned is another keyword, it is new keyword that we are introducing now. If the prefix is to an integer type then again the integer type but it indicate type where only non-negative integers to derive, either 0 or positive integers. And instead of unsigned, if you use prefix sign, or no prefix at all, that indicates both negative and non-negative integers can be stored. So for example, unsigned int would denote an integer types, which can store both which can store only non-negative integers. On the other hand sign int, simply int more denotes an integer types which can store both negative as well as non-negative integers. So here are the example which was discussed. Signed int is the same as just int both negative and non-negative integers can be stored. Unsinged int denotes an integer type that can only hold non-negative integers. So we cannot really assign a negative integers (( )) type unsigned int and results in some kinds of error.

(Refer Slide Time: 08:19)



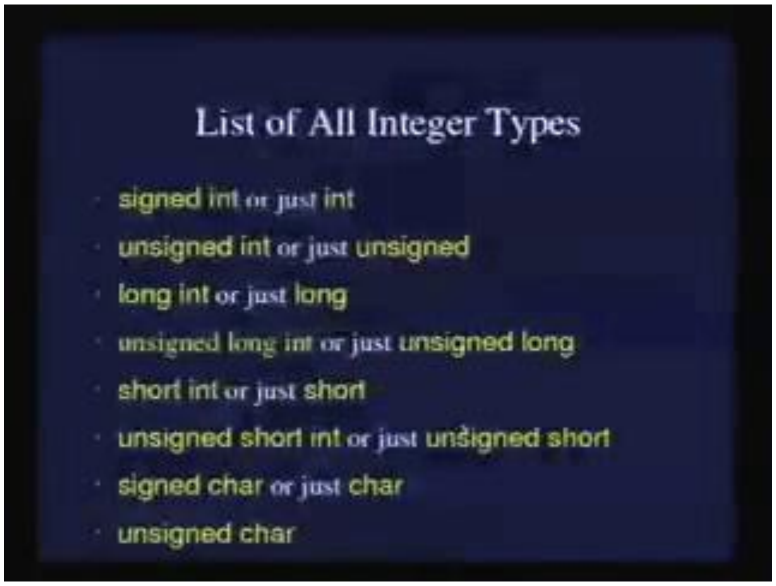
Integers of Various Sizes

- In addition, integer types can be of various "sizes". The size of a type here denotes the number of bits used to store values of the type.
- A type with a bigger size can store a larger range of integers.
- char, short int (or just short), int, and long int (or just long) are the integer types of various sizes in C.
- Each of these can signed or unsigned.

Ok on the other side, integers can also be of various sizes apart from being signed or unsigned. So this size of type here really denotes the number of bits used to store values of a particular type. If you recall from the earlier lecture, we had seen that the integers are stored internally in the binary form. And number of bits available for storing the integers really determine what is the largest integers that can be stored in that particular type.

So a type bigger size can store a larger range of integers. The different sizes integer types in C are char, short int, which can be used as just short, int, and long int – long int just termed as just long, these are the integer types of various sizes in C, we discussed in a manner. And in addition, each of these can be signed or unsigned.

(Refer Slide Time: 09:24)

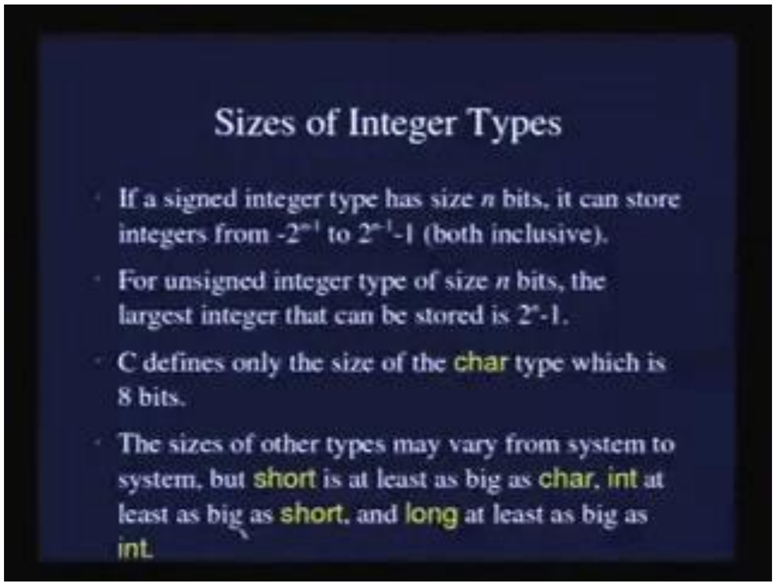


List of All Integer Types

- signed int or just int
- unsigned int or just unsigned
- long int or just long
- unsigned long int or just unsigned long
- short int or just short
- unsigned short int or just unsigned short
- signed char or just char
- unsigned char

So here is the list of all the integers types in C. We can have signed integer, which can be signed or just int. We can have unsigned integer which you can unsigned int or just unsigned. We can have long integer, which you can long int or just long. And unsigned long int or just unsigned long, the short integer, we written as short int or just short. And unsigned short integer is written as unsigned short int or just unsigned short. A signed character which we written as signed char or just char don’t really get (( )) by the fact that the name here is char instead of int, this is for a (( )) which will discuss later on in the talk about representation of characters in computers. At the same type for used to representing a character also, but it actually just another integer type. So, we just think of it right now as a integer type and the unsigned char type.

(Refer Slide Time: 10:39)



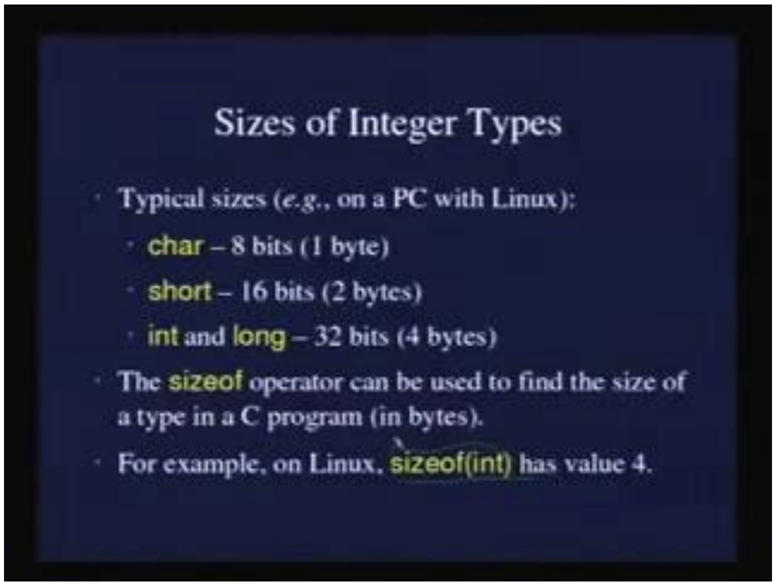
Sizes of Integer Types

- If a signed integer type has size *n* bits, it can store integers from -2^{n-1} to $2^{n-1}-1$ (both inclusive).
- For unsigned integer type of size *n* bits, the largest integer that can be stored is $2^{n}-1$.
- C defines only the size of the char type which is 8 bits.
- The sizes of other types may vary from system to system, but short is at least as big as char, int at least as big as short, and long at least as big as int.

Let’s see what the sizes are. And what the sizes has to with the smallest-largest integers that can be stored in variable of certain types. So if you have a signed integer type with size n bits, then it can store integers form minus two to the power n minus one to plus two to the power n minus one minus one both inclusive. We just state that and not really sure while it is prove. On the other hand, if we are talking about an unsigned integer types again a size of n bits, then the largest integer that can be stored is two to the power n minus one. And of course, the smallest integer is 0. Remember that the unsigned integer type, you cannot store the negative numbers. Now what are the different integer types of various sizes that we have seen so far, char, short int and long, the C stands that definition, the C language definition defines only the size of the char type which is eight bits, which is if you recall same as one bytes, a group of eight bits is called a bytes.

And it doesn’t define the sizes of the other types like short, int and long. And so the sizes of these types may vary from different computer to different computer, from one computer to another. So on one kind of a computer the short may be 16 bits, on another it might it just eight bits and so on. But what is the language guarantees is that short is at least as big as char, int is at least as big as short, and long is at least as big as int.

(Refer Slide Time: 12:14)



Sizes of Integer Types

- Typical sizes (*e.g.*, on a PC with Linux):
  - char – 8 bits (1 byte)
  - short – 16 bits (2 bytes)
  - int and long – 32 bits (4 bytes)
- The sizeof operator can be used to find the size of a type in a C program (in bytes).
- For example, on Linux, sizeof(int) has value 4.

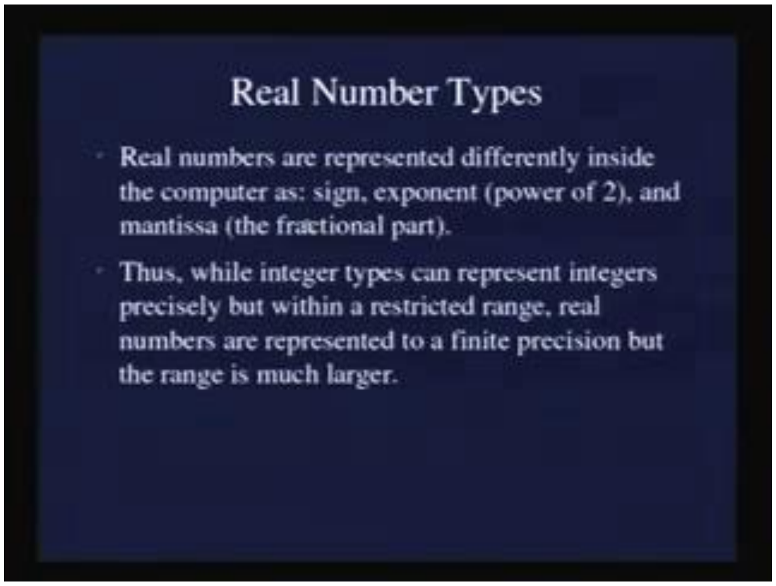
So one a typical PC, which is running Linux, which is the kind of machine that we are using in this course, these are typical sizes of the various types. Char always of course as eight bits, remember that char is 8 bit on every machine. And on a PC with Linux, short is 16 bits or 2 bytes; int and long are both 32 bits or four bytes. So on this particular machine, it doesn’t really matters whether a (( )) variable as int or long because both are 32 bits. But remember on different machine, it might make a different, because on a different machine, for example, int could be 32 bits and long could be 64 bits or may be int 16 bits and long is 32 bits.

Therefore, we have to use the characters, choose the characters of variables, depending on how large we think, the value of that variable is (( )). Now the sizes of these types can vary from system to system, whereas the way of C program to find out what is the size of given type within the program itself and that is the using the size of operator in C. So if for example, if I use a special type of int then this is have the value four on a (( )) Linux, so we can just illustrate this in a small program.

Here is the small program which (( )) sizes of various integers types. So note that this program has no input, it doesn’t take any inputs from the user, just int 4 print f statements and each line that size of n, particular type is so many bytes. So this percent d would be replaced by the value of the parameter, which in this case, would be size of char, which is of course going to be one byte. And in this case, this is replaced by size of short, size of int and size of long and so on. Note that the size of unsigned short is the same as short and similarly the size of unsigned int is same as the size of int and so on. So we don't really need to find out sizes of unsigned d and signed (( )). So let's compile this program and see what output it gives.

This confirm the fact that I just told you and as from this particular machine, this kind of a machine, which is a PC running a Linux the char has size one bytes. Which you remember for every system but on the system short has size two bytes, and int and long has size of four bytes which may be different on different system. But you remember that the guarantee of common C language is that the short will have the size at least as much as char, int has size at least as much as short, and long will have size at least as much as int.

(Refer Slide Time: 15:47)



Real Number Types

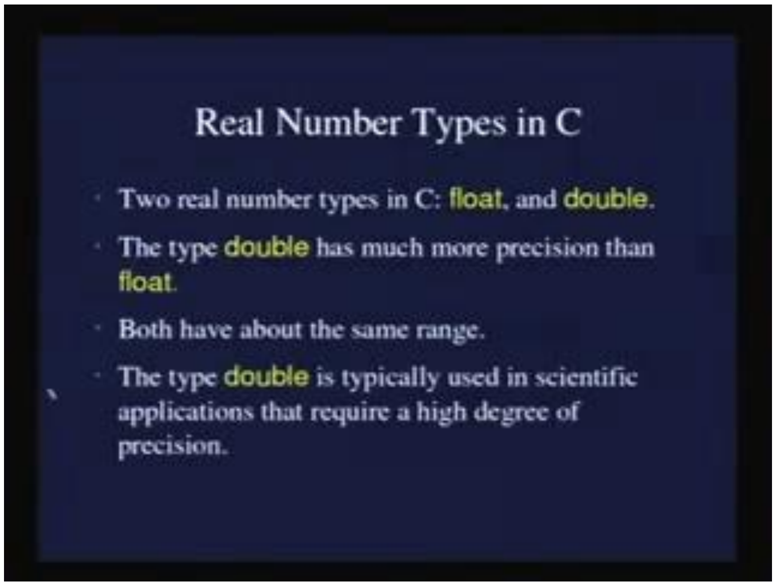
- Real numbers are represented differently inside the computer as: sign, exponent (power of 2), and mantissa (the fractional part).
- Thus, while integer types can represent integers precisely but within a restricted range, real numbers are represented to a finite precision but the range is much larger.

Now let's go back to another type that will be quite useful, and that is the type of real numbers. So we have been dealing only with integers in the program that we have written so far, but as you can imagine often times you might to deal the real numbers, especially when you want to do scientific computation. And real numbers are represented very differently inside the computer form an integer. Real number is really represented as sign whether the number is positive or negative, and exponent which is some power of two. And mantissa are the fractional part of the integer. We don't really need to know the idea behind this in great details. Point that we are trying to make a responding time, that is the representation of integers is very different from char that is for real numbers.

And implication of this particular representation is that real number can only the represented approximately that is we have only a finite number of bit to represent the mantissa or the fractional part of real numbers, and so the accuracy is limited. On the other hand, the integers has absolutely precise. But the difference is while the integers types has relatively smaller range so for example, on the machine that we have saw the size of integer type the four bytes and thirty two bits, so that means that the largest integer that we can store int type would be two to the power thirty one minus one. The range of real numbers on the other hand is much larger, but the precision is restricted.

Next now look at the real types at available in C. We have two of them really, they are called float and double. The word float comes from the fact that the representation of real numbers that we just saw is the floating point representation; and the word double comes from the fact that the double types of about twice the precision has (( )) float type. The type double has much more precision than float, but both have about the same range, the smallest and largest, real numbers that we have stored are about equal in both these types. The double type is typically used in scientific applications that require a high degree of precision.

(Refer Slide Time: 17:35)



Real Number Types in C

- Two real number types in C: float, and double.
- The type double has much more precision than float.
- Both have about the same range.
- The type double is typically used in scientific applications that require a high degree of precision.

Let’s now write a small program to illustrate the float type. And this program is to compute the area of the circle, really a simple program, give the radius of the circle. So obviously, we want to represent both the radius and the area of the circle by real numbers. Here is the program, remember that hash include the stdio dot h will because (( ))pretty much all programs are derived, this (( )) we are going to use some standard input and standard output, function in the standard (( )). So here are the variable that we are going to use. The variable radius will represent the radius of circle that user will input. The variable area will represent the computed area of a circle PI of course is a well known mathematical constant, and here using that value three point one four one nine. Of course, we could have (( ))value for pi.

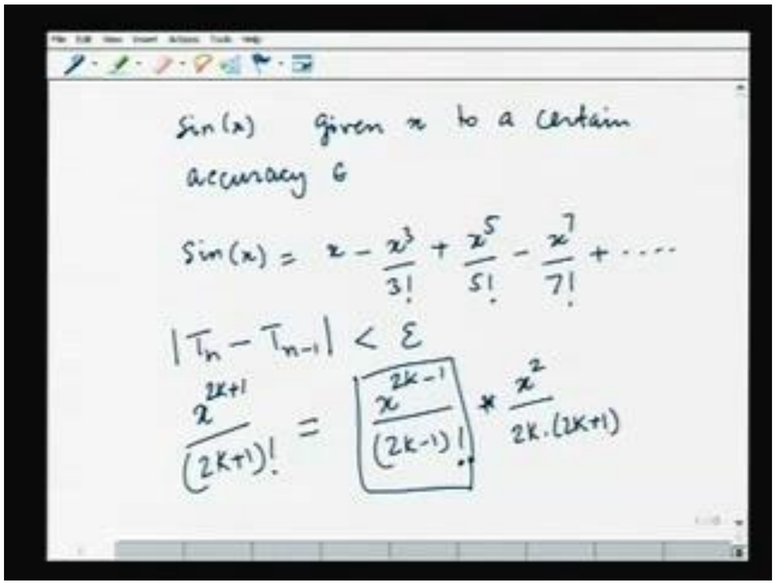
So this program is quite simple and easy, you can see the enter we would prompt to enter the radius, and then we have to read the radius from the user. And then now that the radius floating point variable, or a float of f, we cannot use scanf for (( )), still scanf will read the value of radius, but instead of using percent d, we will use percent f. And remember that and scanf the variable have to be perceive by the ampersand. So once we know the value of radius, so the value of area is very simple to compute it, pi r square, pi times radius times, radius and then we are printing the output which is the area of the circle with radius so much is so much. So again note that to print a floating point value or a real value. We are using percent f, percent d for printing the integer. And we are using the same print f statement will print two real values along with some text, so we have two percent f in the format string, and the first percent f will displays the value of the radius, and second percent f will displays the value of the area.

So let’s compile this program. Let’s say we give the radius as 2.56 whatever the radius is and that gives us correct area of the surface. (( )) make it 20.59 0 755, which you can is verify using a calculator. One more point I would try to make during the same program, let adding it again and that is in this program the variable pi, is actually (( )) variable in the sense we never want to change it. Once we have given it a value, note that one more using that real program is the variable has (( )), the same type has when it has been declared. This is the first time we have been doing this. So float pi is equal to 3.1419. So we can left the initialization of the variable, which is the declaration.

As I said really don’t want to modify the value of pi ever in our program. So to indicate this we could (( )) this declarations by the (( )) form. The keyword for the complier that this variable is really constant which would not be modified by the program, so it only be initialized along with the declarations. It cannot be given a value later on. So in the point of doing this is that if (( )) by mistake for program happen to modify the value of such variable, the complier will give a error message. Let’s try to do that actually. Let’s say 5.56. So this will give an error.

Assignment of read only variable pi, so this is not permitted, so that is advantage of declaring variable as constant when we don’t really want to a ever modify their values. So this program compile is ok and well works same as before. Here save done. So let’s end this lecture with a problem for you. The problem is to compute the value of sin x given the value of x to a certain accuracy, let’s say sin x which is also given.

(Refer Slide Time: 23:25)



Sin(x) given x to a certain accuracy ε

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$|T_n - T_{n-1}| < \varepsilon$$

$$\frac{x^{2k+1}}{(2k+1)!} = \boxed{\frac{x^{2k-1}}{(2k-1)!}} * \frac{x^2}{2k.(2k+1)}$$

Now accuracy basically means that value of sin x must not that is computing by the program must not differ from the actual value of sin x by more than this amount. Now the way you are going to do this, is to use the Taylor expansion for sin x, we call that sin of x is equal to x minus x cube by three factorial plus x power 5 by 5 factorial minus x power 7 by 7 factorial and so on. So we have to use this Taylor series for compute the value of sin x, given the value of x, and since we have to compute for the accuracy for epsilon (( )) you should stop adding the terms of sequence once the absolute difference between last two terms that has been computed is less epsilon.

So let’s take T n is the last term that we have computed and T n minus 1was the term that we computed. Before that then the difference is less epsilon. The absolute difference in less than epsilon then we can stop computing. And the sum that we have accumulated so far is the value of sin x, so that the accuracy. Also note that you can in every iteration, you want to compute some x to the power 2 k plus 1 by 2 k plus one factorial. We don’t need to the same x again to the k plus 1 and k plus 2 factorial. You can use that fact in that previous iteration use, you computed the term x power 2 k minus 1 divided by 2 k minus 1 factorial. So given this value, to compute the next term which is this all you need to do is, you need to multiply this by x square and divided by 2 k times 2 k plus 1. So you can see this will become x to the power 2 k plus 1 into 2 k plus 1 factorial and that will give you the value of the next term. So try to solve this problem and write the C program for computing the value of sin x, will the solution to solve the problem in learning of the next lecture.