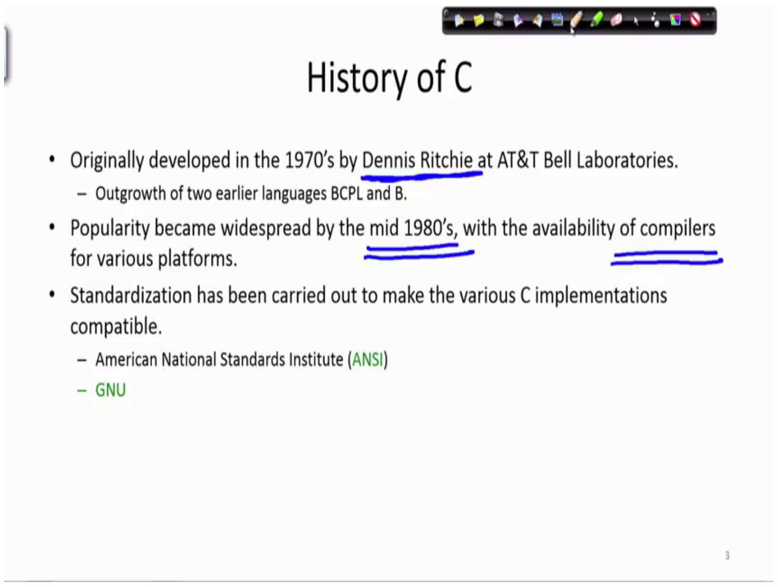
Problem Solving through Programming In C

Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 07
Introduction to C Programming Language

So, we have started our discussion on a specific programming language; that is C and once again I repeat, that a C program will be constituted of some valid words of the C language, just as the an English sentence must be constituted with valid English words, similarly for C and must be every language has got a grammar. So, C has got a very strict grammar and any statement that is written in C that is not adhering to that grammar will not be accepted by the compiler. The compiler will reject it, saying that it is a syntax error or grammatical error. So, before going further into discussions on C briefly, let us look at the history of C, it was originally developed in 1970 by Dennis Ritchie all right and his book.

(Refer Slide Time: 01:17)



History of C

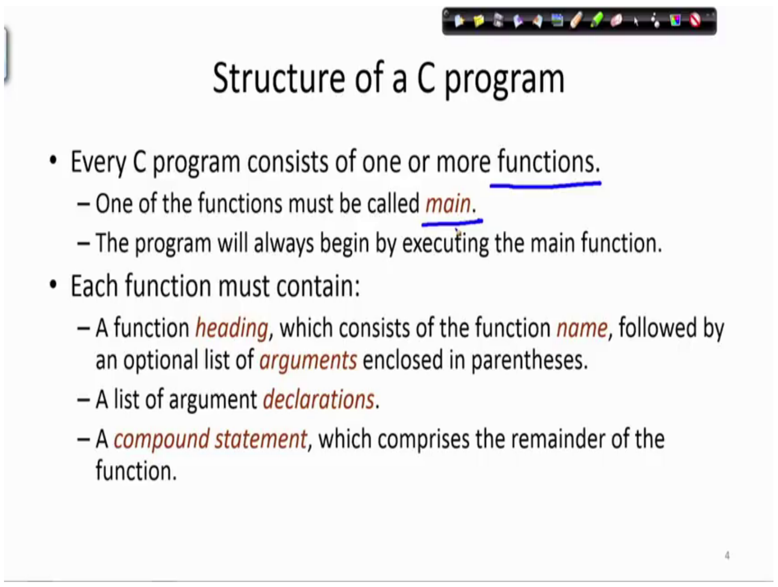
- Originally developed in the 1970's by Dennis Ritchie at AT&T Bell Laboratories.
  - Outgrowth of two earlier languages BCPL and B.
- Popularity became widespread by the mid 1980's, with the availability of compilers for various platforms.
- Standardization has been carried out to make the various C implementations compatible.
  - American National Standards Institute (ANSI)
  - GNU

3

I have also referred to you at AT and T, Bell labs and become very popular by the mid 1980's, because it was the compilers. Now, a language, a computer language cannot be popular, cannot be used unless there is a compiler for it. So, it took some time, although it was developed in seventies, but many compilers in, for different platforms, for workstations, for PC's, for different platforms were made available by mid 1980's and this became very popular and there were some standardizations that also took place.

Now, even that we, let us look at the structure of a C program. Let us look at the structure of a C program, here every C program will consist of

(Refer Slide Time: 02:15)



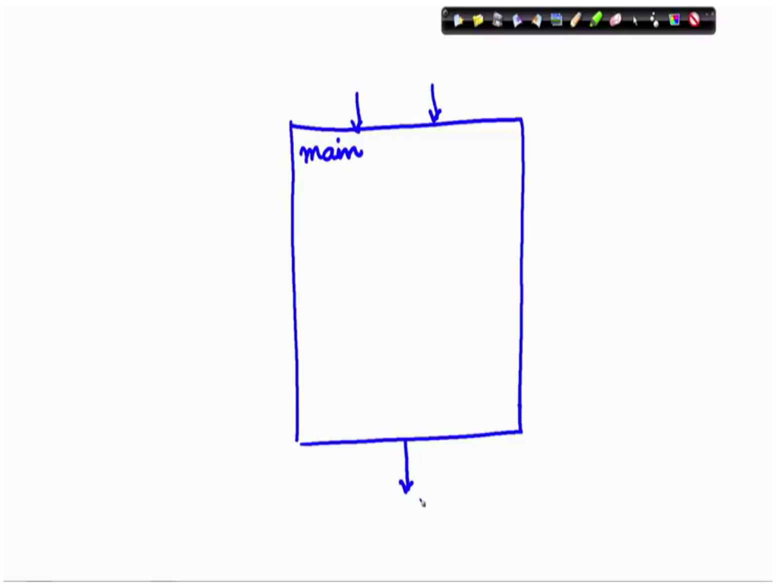
Structure of a C program

- Every C program consists of one or more functions.
  - One of the functions must be called *main*.
  - The program will always begin by executing the main function.
- Each function must contain:
  - A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.
  - A list of argument *declarations*.
  - A *compound statement*, which comprises the remainder of the function.

4

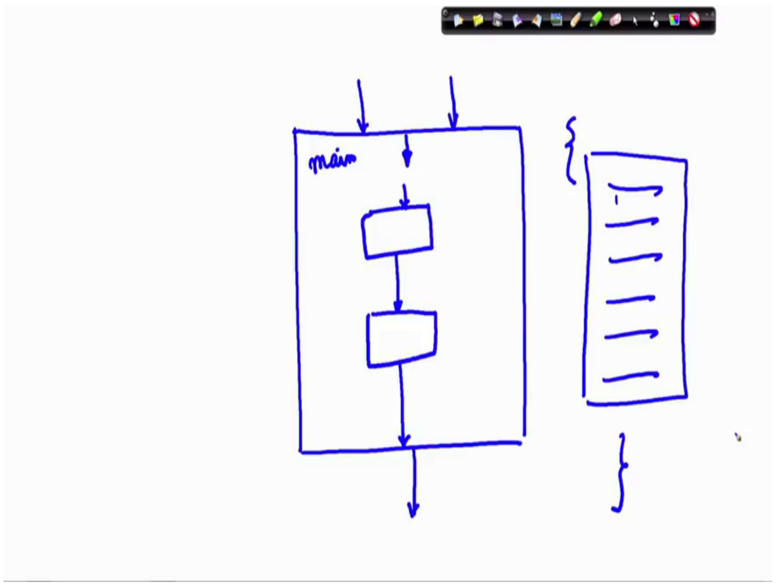
One or more functions one or more functions. One of the functions will be given the named main, which is the function. Now, I in the last lecture, I said that there should be at least one function, if nothing else is there, if the even, if the program is. So, simple that I do not need any sub machines, in that case I can do it with only one simple function and that function has to be named main and if there be a number of say for example, sub func, number of functions or sub machines, in that case also there has to be one program, which has to be called the main and the rest can be given some other names, the program will always begin by executing the main function. So, once again let us go to the diagram that we had used in the last lecture.

(Refer Slide Time: 03:17)



Say I have got a program, which has got only one function, very simple then this function will be called main and no other sub functions are there it will take some inputs, whatever the inputs are, it will process that and the output will be made available. Now, suppose I have got two sub machines, required for this one here and after that another one.

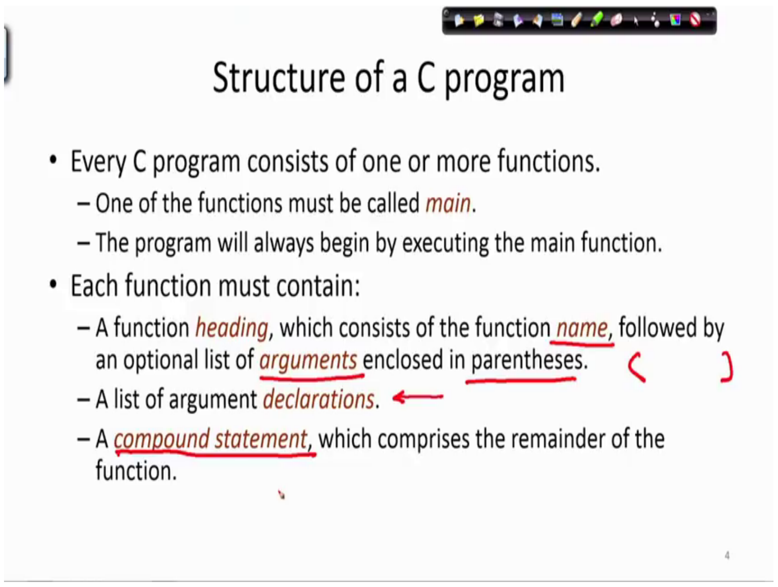
(Refer Slide Time: 03:46)



So, this sub machine will do something, then this sub machine will do and then will come out and this is the output and there is some input coming in. Here these are the inputs and they are some sub machines, but I have got one machine, which is the main. This is again the main. So, although I am taking help of the sub machines, I have to first enter the main function and then from the main function I can enter here and go somewhere else and ultimately, I will have to come out through the main function. So, we cannot escape the function.

So, the main will be there and typically we write the main any C body, within two curly braces, here whatever I am drawing as a diagram that is equivalent to these two curly braces. Inside this, whatever is written is a C program, whatever I write here, whatever C program I want to write, I have to write within these two curly braces, I will come back to this in a moment. So, so there

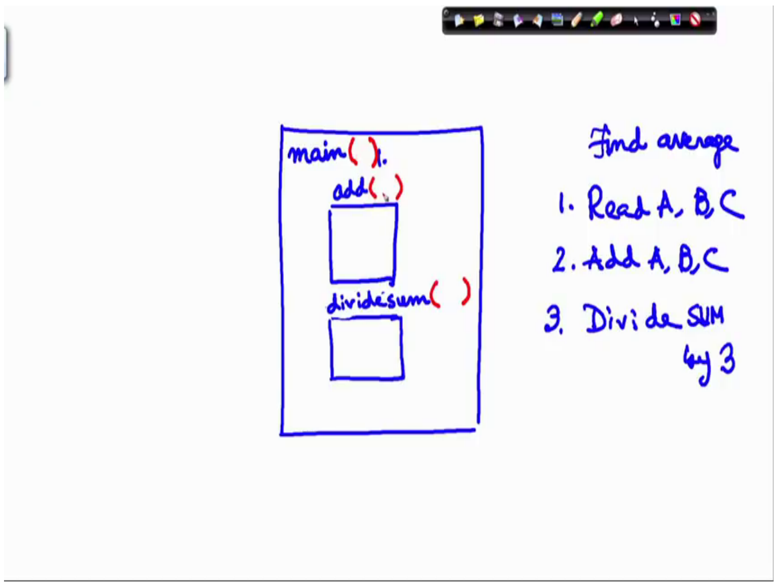
(Refer Slide Time: 05:10)


Structure of a C program

- Every C program consists of one or more functions.
  - One of the functions must be called *main*.
  - The program will always begin by executing the main function.
- Each function must contain:
  - A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.
  - A list of argument *declarations*.
  - A *compound statement*, which comprises the remainder of the function.

has to be one function called the main and the program will always begin by executing the main function, each function must contain a heading, which consists of the function, name followed by something application, say; followed by an optional list of arguments, I will explain that later.

(Refer Slide Time: 05:49)

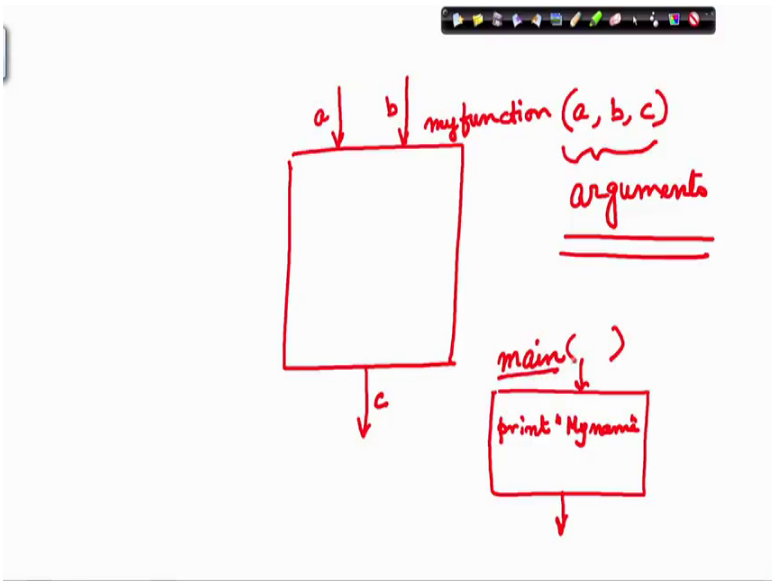


Let us again come back to this, that I have got a function, I have got a program which has got one function, whose name is main and I have got another function, whose name is say add and there is another function whose name is find. Is it readable? Find average or let me, this is a little confusing, let me rename this, I name it as divide sum, take the example of our simple finding average all right. I want to find average.

So, what are the sub tasks, one is I have to read the numbers, say read say ABC that reading suppose, I am doing here this read ABC is here, one after that I am doing, add ABC, say adding ABC is being done by this sub machine say, and therefore, I have to give a name to this, sub machine that this sub machine is inside the body of this sub machine, I will actually do the addition, but the name of the sub machine should be expressive of what.

This sub machine of the function is supposed to do similarly, after I do that then I divide sum right I will divide sum by 3 by whatever 3 here in this case. So, that is being done by this sub machine this is too simple these are two simple sub machines, but this is being done by this submachine. So, they each of them has to be given some name each of the functions including main has to be given some name. Now, along with that there is another point that has been mentioned here, there is the argument each of these functions must have a place, where I can write the arguments right. Now, I am not writing anything on main, but say when I say what are these arguments, if I just consider this separately all right, any function separately any function

(Refer Slide Time: 09:05)

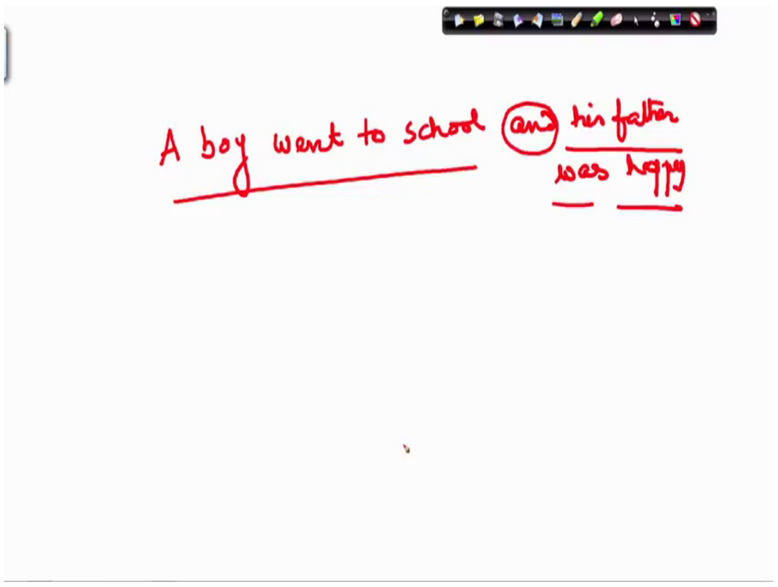


If I consider separately, now this function will have some inputs and some outputs. What are the inputs and what are the outputs? Suppose, the name of this function is my function, then this function will have some arguments, which are given in within this parenthesis and suppose, these are A and B are two variables, which are read by this and C is a variable that is output by this.

So, the name of this function should be associated with the names of the variables, which are taken as input or is supplied as output by this function these are called arguments, we will come to this when we discuss some other very important properties of functions, but for the time being just remember that a function must have a name and there should be some place for writing the arguments, often you will find that a function like main.

Suppose, I am writing a function which will simply print a particular line, then inside the body of the function I just want to print, I will say print my name, is something whatever. So, my name just say my name. Now, this function whenever it is entered, it will just print my name it is not requiring any values to be passed, but still then I will have to put this parenthesis with the name although, I may keep the inside of this parenthesis gland. So, a function in order to be a valid function in C, these are a C rule that in order to have a valid function. A function must have a name and a place for the arguments. So, a function heading is the function name all right, followed by an optional list of arguments enclosed in parenthesis like this is optional, because it can be blank and there are these are the things that there will be some argument declarations, etcetera, will come to this particular part later, but before that we have to understand a compound statement. What is a compound sentence? Say, simply in English if we consider, you know that a simple sentence has got only one principal, verb principal finite verb; however, I can have a compound sentence like a boy went to school and his father was happy say.

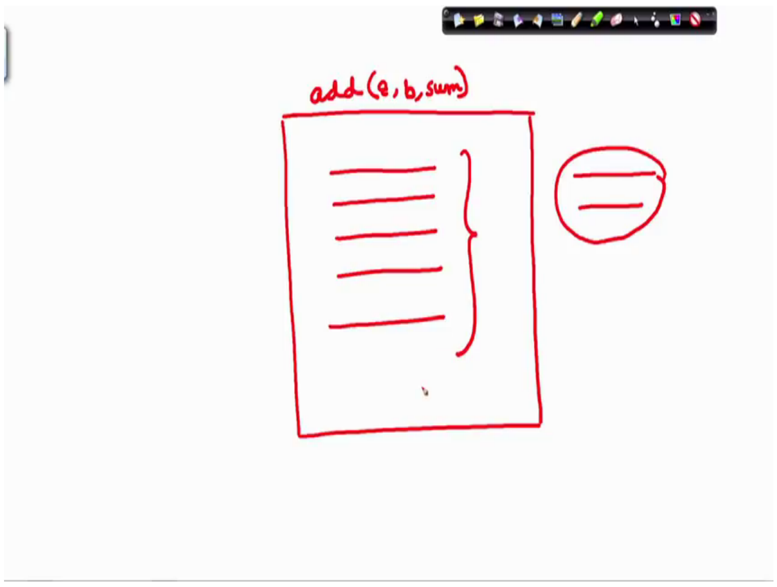
(Refer Slide Time: 12:41)



So, here there are two sentences his father was happy and a boy went to school. These are two simple sentences and we are connecting them with an end here, all right. So, these are compound sentence. Similarly, we can have complex sentence and all those.

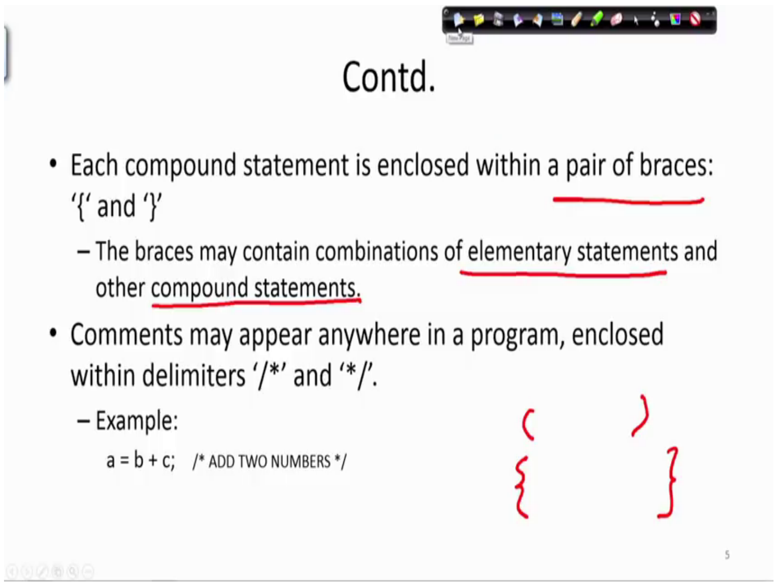
Here the idea is similar, but not exactly the same, I am saying that.

(Refer Slide Time: 13:23)



A function is a machine, which has got a name like say, add and sum parameters A B C A B and the third parameter is sum; that means, it will take A as input B as input and will produce sum, but here I write a number of C statements, a number of C statements are written over here. Each of this C statement is a statement and all these together are together forms are compound statement. Whenever, we find more than one C statement working together that will be a compound statement. So, here the entire function add will be if it consists of a number of statements, then it is a compound statement, in the default case, in the very trivial case, when the program consists of only one statement then also we can call it a compound statement, but with only one statement.

(Refer Slide Time: 14:55



Contd.

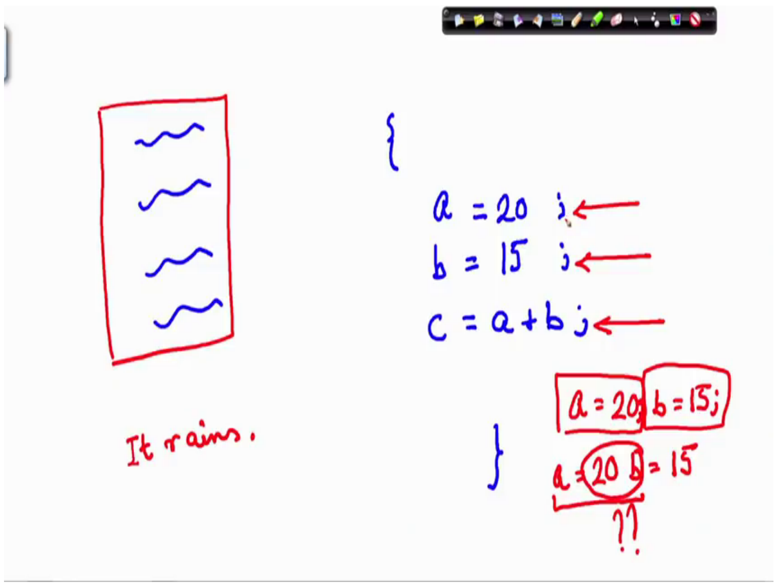
- Each compound statement is enclosed within a pair of braces: '{' and '}'
  - The braces may contain combinations of elementary statements and other compound statements.
- Comments may appear anywhere in a program, enclosed within delimiters '/*' and '*/'.
  - Example:

```
a = b + c;    /* ADD TWO NUMBERS */
```

So, next, each compound statement is enclosed within a sorry, is enclosed within a pair of braces like this, this is called braces, this is called parentheses and this is called braces right.

The braces may contain combinations of elementary statements, a single one or other compound statements. Now, once again let us see till now, I was drawing the machine as this

(Refer Slide Time: 15:39)



I was drawing it as a rectangle. Now, I will move towards, more towards C. So, I will say that this boundary will be specified in C as this sort of boundary, whatever is here is within this parenthesis. Now, each of them, each of this braces, I mean inside these braces, we will have some statements. It can be an elementary statement like say, A assign 20 or maybe more number of statements. Now each of the C statements are delimited by a semicolon.

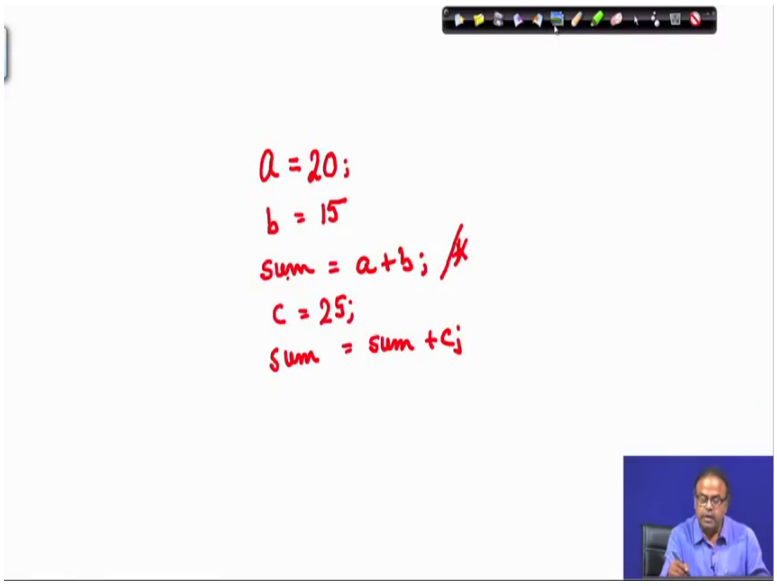
This is very important, this you must remember, each of these are delimited by semicolon, if I do not write the semicolon, the sentence is not completed, just like in English we have to write a sentence and we have to complete it by a full stop right. It rains very simple sentence, it rains and then there is a full stop here.

Similarly, in C, the end of one single elementary statement or a set of statements is pointed out by semicolon all right. Otherwise, it will be ambiguous, there can be a problem like for example, if I write A 20 B 15 and I forget the semicolon. If I can, I am writing one after another, because that is nice to write one after another, but if I write them side by side that is also equivalent, but if I put semicolons here, the compiler will understand that this is a one statement.

This is another statement, but if I do not give the semicolon here, for example, I write A 20 B 15, then the compiler will be in a problem, because it does not know what it is supposed to assign 20 or 20 B or whatever is it possible to assign 20 B many things will come. So, we must be very careful about completing the statements with semicolon.

So, the braces may contain combinations of elementary statements and other compound statements. Now, there is another very important thing called comments, whenever you write programs, you must be generous of writing comments. Now, what are these comments? Comments are statements, which are not, which are not converted by the compiler to machine language, but then why do we write it? The reason is that whenever we write a complicated or a large enough programs, the presence of the comments enables us or enables suppose, I have written a program, it will enable somebody else to understand the program. So, if I write something like say

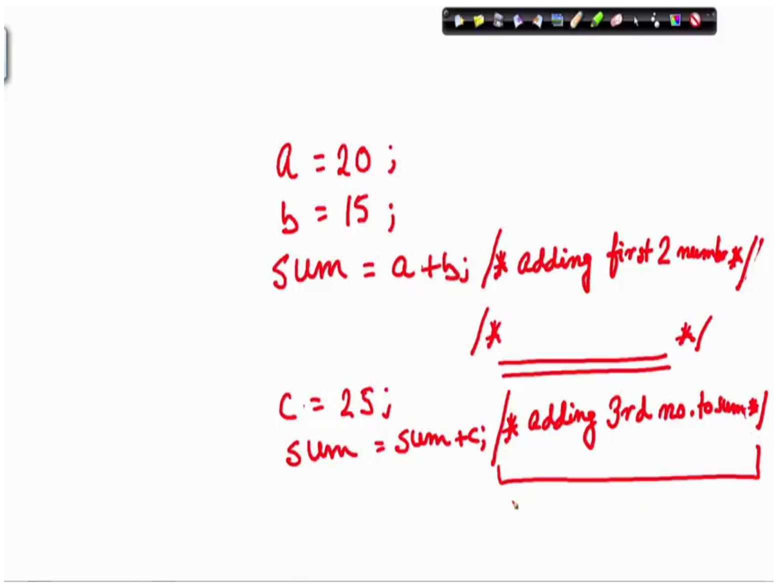
(Refer Slide Time: 19:45)



```
a = 20;
b = 15
sum = a+b; /*
c = 25;
sum = sum +c;
```

I am again, I am not going to a complicated example as yet I am remaining with that finding the average of the numbers, but suppose, I am doing it in two steps all right. Say, I do, I am writing, I am just doing something like A assigned 20, B assigned 15, sum assigned A plus B and then C assigned 25, sum assigned sum plus C. Now, when I write this, somebody may this rather too simple, but somebody may say what are; what am I doing here at both the places? I am doing sum. So, that somebody else does not get confused, I can write something like this. Let me just check the syntax here, yes I was right. So, what is happening?

(Refer Slide Time: 21:07)



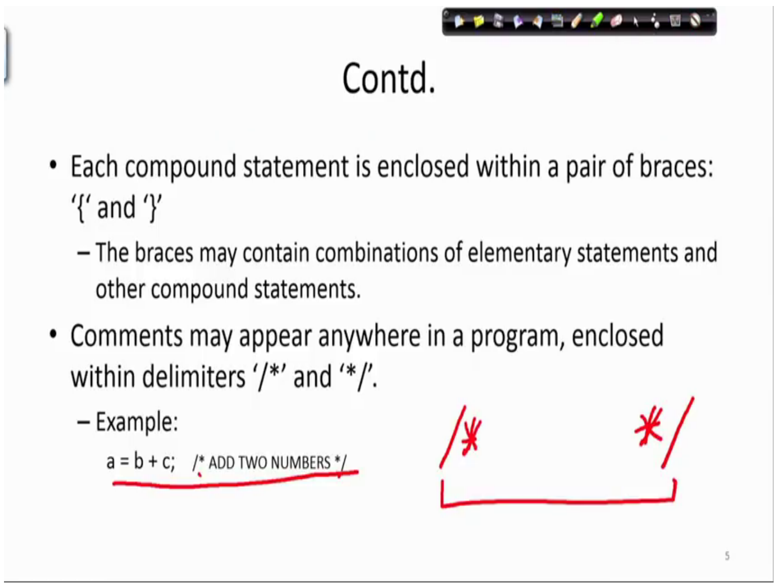
```
a = 20;
b = 15;
sum = a+b; /* adding first 2 numb */
           /* ________ */
c = 25;
sum = sum+c; /* adding 3rd no. to sum */
```

So, I can write say, A assign 20, B assign 15, note the semicolon that I am putting after every statement, sum is equal to A plus B. A plus B and here I can put in a comment, say adding first two numbers I am ending this with this symbol, this is again I am doing a bad job here. Say, I write numbers and then I put this sort of symbol.

So, here you see I put here this and here reverse this, whatever I write in between that is assumed by the compiler to be a comment statement. So, the compiler need not convert this into machine code. So, here again I can write later say CS sorry, C assigned 25 and sum equals some plus C. Now, I can explain that adding whatever I want I can write in any form, adding third number to sum and put this I am sorry, here always I am making, I am not being able to manage the space, it should be, there should be a space in between. So, I will have to rank the third number to sum and I put this end of comment. Now, this part will not be compiled.

So, for any program, when we write as the program, becomes more and more complicated. We should be generous about writing the comments that will also help us. Say, you have written a program today and he want to look at it after, say one month and see what you did? Such comments will be helping you to understand what you did or somebody else of course, to understand what you did right. So, for examples let us see here,

(Refer Slide Time: 24:10)

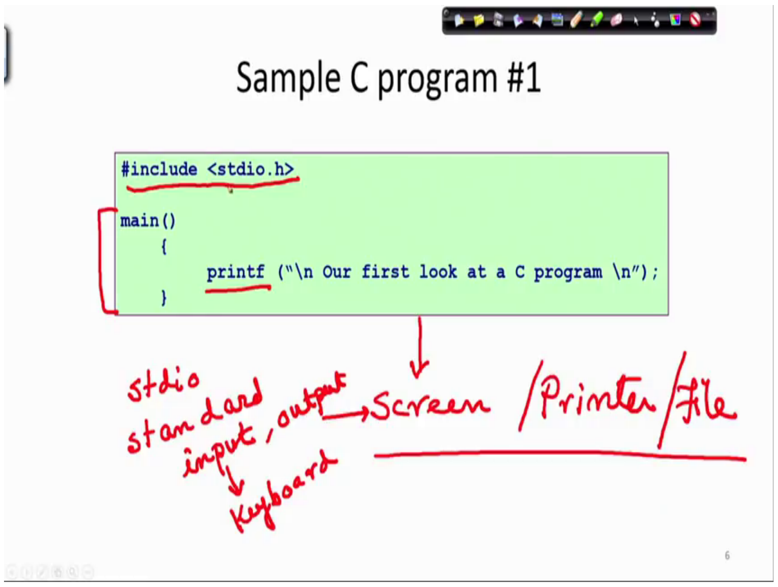

Contd.

- Each compound statement is enclosed within a pair of braces: '{' and '}'
  - The braces may contain combinations of elementary statements and other compound statements.
- Comments may appear anywhere in a program, enclosed within delimiters '/*' and '*/'.
  - Example:

```
a = b + c;   /* ADD TWO NUMBERS */
```

I have got a simple thing A assign B plus C. So, here you see, add two numbers between this to this and the end of comment is this, all right.

(Refer Slide Time: 24:30)



Sample C program #1

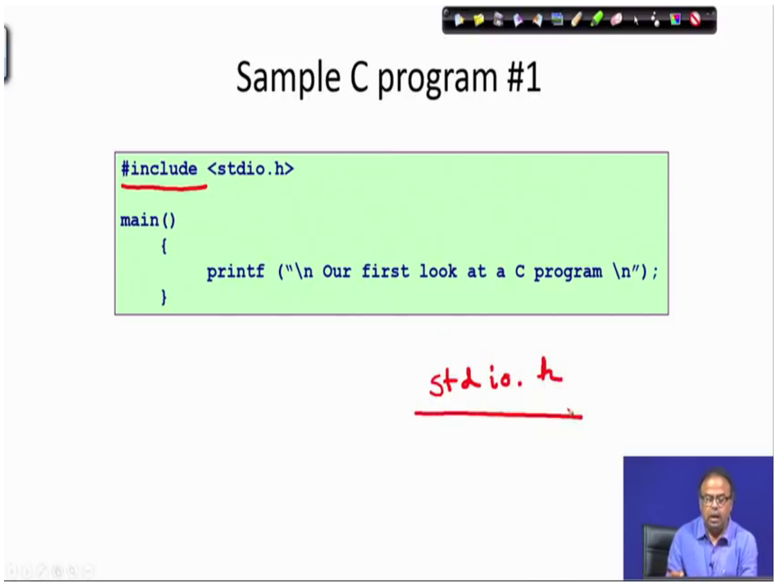
```
#include <stdio.h>

main()
    {
        printf ("\n Our first look at a C program \n");
    }
```

Now, let us look at a simple C program there for now, this part I will be explaining separately, but before that let us look at this part of the code is a very simple program, simplest possible program. What it does? It simply prints our first look at C program, this line as is will be printed as is will be printed, where will it be printed this? Will be printed on the screen or on the printer or on some other file, where will it be printed here, we are writing another statement hash include stdio dot h.

This stdio dot h means standard input output std for standard. So, it is stdio is standard input output. Now, if nothing is specified, then the standard input is our keyboard, by default. It is the keyboard and the output by default, is the screen. So, when I write this then, when I write print f; that means, this line will be printed on the screen that is the meaning of a stdio dot h.

Now, this single piece of simple program has got many things to illustrate. This term hash include is an instruction to the compiler that you need not convert it to the machine language, but before you convert the program to the machine language, please do this. What is doing this?

(Refer Slide Time: 27:17).



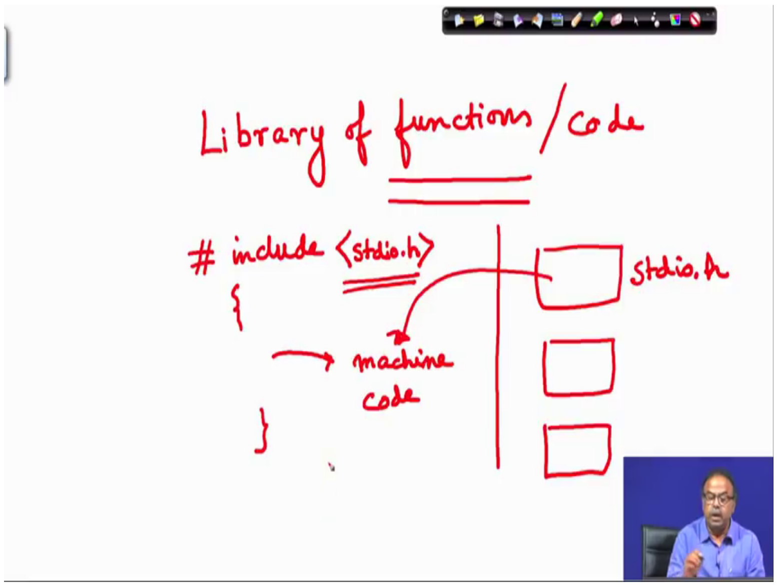
Sample C program #1

```
#include <stdio.h>

main()
    {
        printf ("\n Our first look at a C program \n");
    }
```

stdio.h

Please, include stdio the standard i o dot h. Now, what is this stdio dot h? In order to understand this, you have to know what a library is now? Any programming environment, any programming environment provides you with a library of functions C, provides you a library of functions.

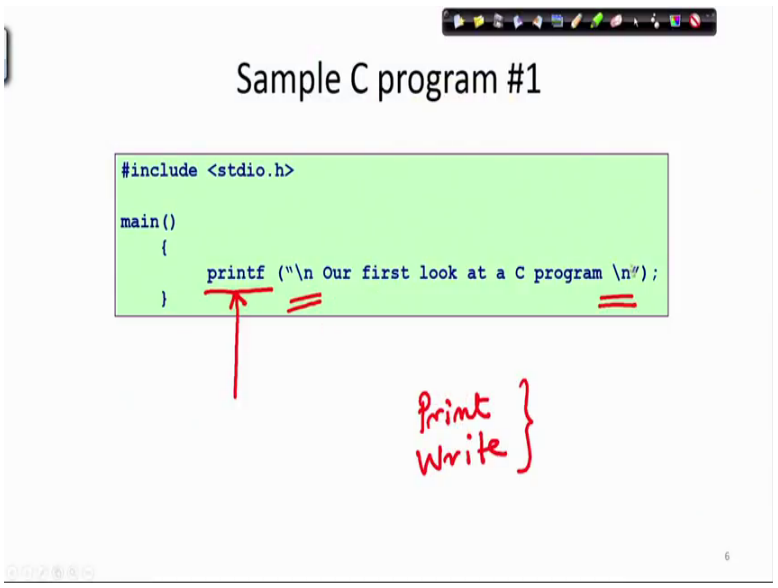
(Refer Slide Time: 27:36)



Now, this library of functions and library of other packages, which are library functions or codes, let us see that is already there now. So, I am going to write a program here and here there is a library and in that library there are many things here, all right and one such thing is stdio and all these libraries files have got an extension dot h. Why that is? That we will see later. Now, when I am writing this part of the program, if I say hash include within this corner brackets stdio dot h; that means, whenever I am writing this program, whatever program I have written, that will be converted to the machine code that will be ultimately converted to the machine code.

While doing that, this before, doing that this stdio part must be included or must be included some of the information, must be included, that will tell me that well you see whenever you are doing this print f that print f will be, will have to be done on standard output and what is that standard output? The printer, it is a printer, so, that is the purpose of hash include. There is a preprocessing statement, compiler a preprocessing statement. Now, I had said that every language has got, some of it is own vocabulary. Now, in English when you understand, whenever I write print or write this words, carry meaning to you in English in C, we write it as print f.

(Refer Slide Time: 29:47)



Sample C program #1

```
#include <stdio.h>

main()
    {
        printf ("\n Our first look at a C program \n");
    }
```

This is a special word of C, why they say if there is a meaning for that. That is in C, wherever we write, whatever we write, we consider that as if we are writing in a file, say in your office or in your work desk, whenever you write something, you write it on a piece of paper and put it in a file in C. We consider every input or every output device to be a file and this f stands for file as if I am printing on the file, which file? The printer file, then the next class, we will start with this and will describe some of the other he brew like statements, here which will from, which we will go ahead.

Thank you.