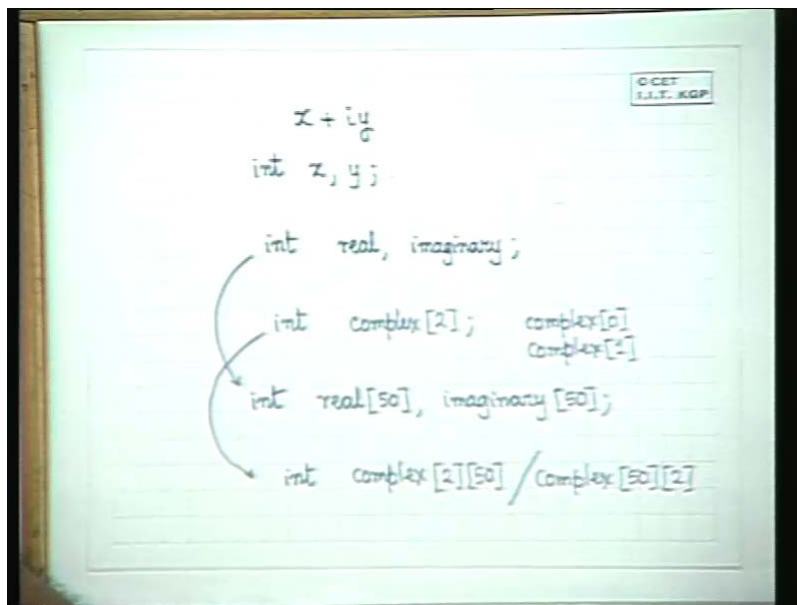


Programming & Data Structures
Dr. P.P. Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 14
Structures I

Hello and today we will start a topic of studying some C language features related to structures. Now structuring or encapsulation is at the heart of data organization or data structuring. For example suppose we are interested in keeping information, a simple information regarding complex numbers and we would like to store complex numbers, read in complex numbers and then we would like to operate on them perform addition, subtraction and even other type of operations. For that we know the normal syntax that we have already studied in C, we would have to define for every complex number we would have to define its real part and imaginary part because the complex number is written of the form x plus iy . Now in order to define a complex number, we would require to define 2 integers in our normal programming language.

(Refer Slide Time: 04:22)

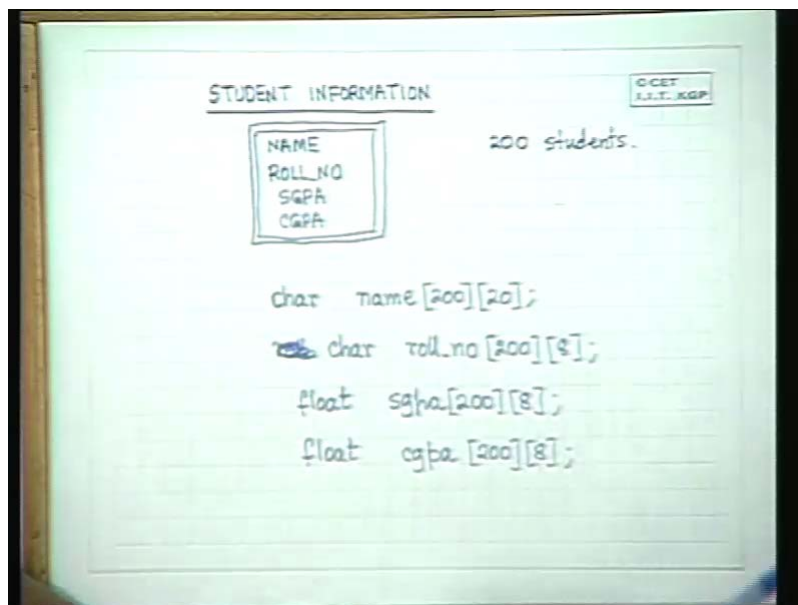


So in our normal whatever we have learnt or seen till now, we would have to define `int x` and `y`. And whenever and we would have to remember which of the values is the real part and which is the complex part. So we would possibly, this is one way of doing it or to make the names much better, we would write `real` and then `imaginary`. The other option which people will choose would be to define a particular complex number as an array of two integers in which the complex zero is the real part **is the real part** and complex imaginary, the complex one is the imaginary part.

If you use arrays, this is one option we can do and if we want to define a set or an array of complex numbers then this would be a two dimensional array and here, so the two options if we take this option to define an array of complex numbers here we would have to define in this situation we would define a real array say 50 and an imaginary array 50 and use the indices to indicate which is the real part of this number and the complex part of this number. And in this situation we would, these are the alternatives either 250 or 52 the other option is these are the two ways in which you can define the complex numbers. And in order to operate on them you would have to operate on these individual operation, each individual data.

Consider another situation where we have to maintain some student information. In order to maintain some student information, we need to keep various information about a student.

(Refer Slide Time: 08:20)



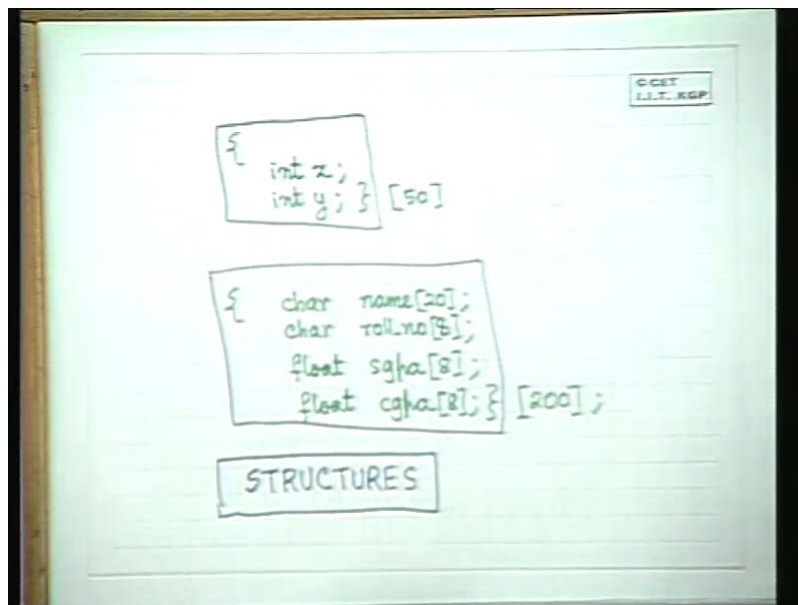
For example let us assume that we keep the name of the student, the roll number of the student. We keep their sgpa's for the relevant number of semesters and we keep the cgpa values for the relevant number of semesters. Now let us see how we would go about doing this. Now let us say we have got approximately 200 students then we would individually define name to be an array of character strings. Isn't it? So we would define name something like this. It would be 200 students and each would be an array of character strings say 20.

Roll number would also similarly be something like this. Now we would have to maintain sgpa values for 8 semesters and equivalent cgpa values for 8 semesters in this situation. Then we would have to define float sgpa 200 8 and float cgpa 200 8. So this is how we can get 200 students information and put them together in whatever we know.

However let us go back to what we are doing. Normally when we talk of structuring of information, we are interested in maintaining student information. So we would like to somehow conceptually keep the information of one student in one place. Here we have kept it in separate arrays and we can access the indices of arrays to find out what is there for each student. Now in such conceptual structures, we would like to keep information in such a way that it is more conceptually appealing to keep student information. For example if I want to access student, I will have some in structure where I can keep all of them together in some form. And then keep an array of this information. That is what we actually wanted to do. We wanted to keep information regarding in one place. Suppose we could declare in one place name, roll number, sgpa and cgpa in one form and then keep an array of 200 of that then that would help us to conceptually structure the information better.

This sort of structuring has got a name and this name is called encapsulation, data encapsulation. We would like to put all this data in capsule or one model. Now C language, similarly in the situation of real and complex numbers, we would like in one capsule to keep one complex number look like this.

(Refer Slide Time: 11:27)

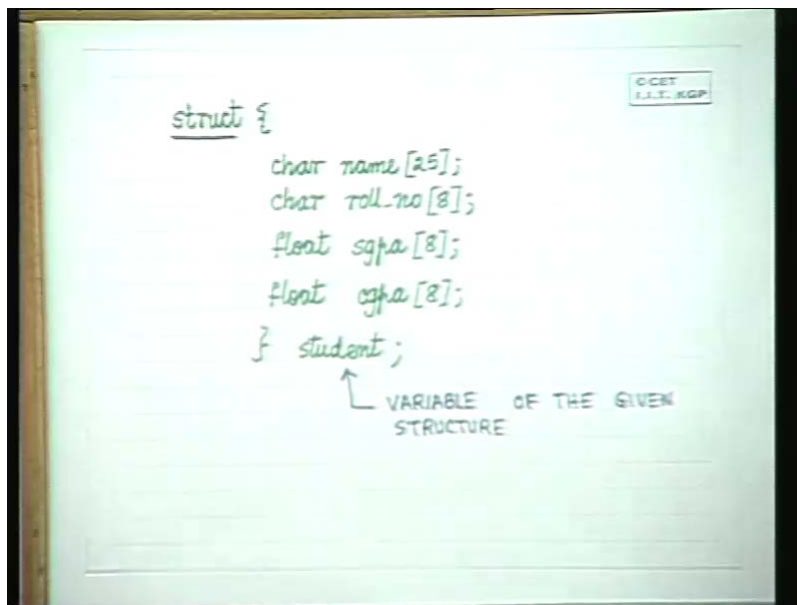


Now I can have an array of complex numbers something like this that is in the situation of complex numbers what we wanted was we wanted an information were we would keep int x int y alright, this would form one capsule. And we would declare an array of size 50 of this, something like this. And in the roll number situation, we would like to define character name say 20, character roll number. What did we have, 8. Float sgpa 8, float cgpa 8 and I have an array of 200 something like this. This is a character array for strings, this can be accessed as strings. These are the, each element is a floating point number for 8 semester.

This is floating point number and so this is sort of conceptual structure which we actually imagine and C language does provide us means of using this information in the form of what are called structure definitions. There is a syntax for defining structures and using elements of structures. So each structure can be defined by a name and this, for example this one, this will get a, this will be a particular type. For example you have integer real character this will be one type, you can even give it a name type like you give an integer a type name.

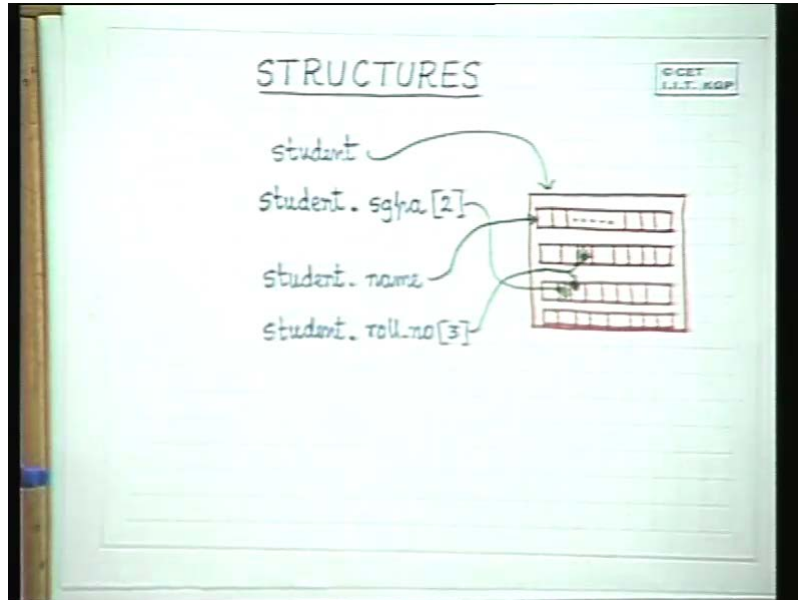
Similarly this would be a type, so we can define several structural types and then define variables which are of that type. So we today in this class we will see how to define C structures and use C structures. This is the one form of defining a C structure, you have a keyword struct and within this to this bracket you will define the structure char name 25, char roll number 8, even you can define a structure as inside a structure, that is no problem, float sgpa 8, float cgpa 8 bracket closed. So this defines this structure and student is a variable, in this format student is a variable having this structure.

(Refer Slide Time: 12:55)



In this format this is a variable of the given structure. Now in order to access any element here say suppose I want to access the cgpa of the third semester that is sgpa of the third semester then what I would do is student dot sgpa third semester 0 1 2 2. If I want to access the name as a string, now this is an array so student dot name will give me the pointer to the start of the array.

(Refer Slide Time: 15:50)

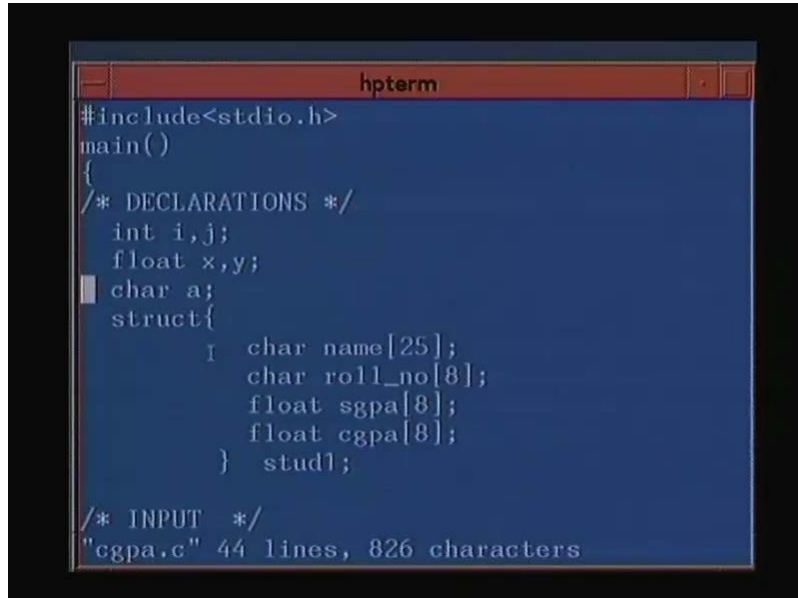


This is the name of the array, so this will give me a pointer to the start of the array. Student dot sgpa 2 will finally lead to the element. Similarly student, what is student? It is a pointer to the structure, alright. It is the pointer to the structure. Roll number is a pointer to the character array roll number in the structure and 3 is the character element there. So if you draw it up, we have got name of 25 elements, we have got roll number 8 elements, we have got cgpa of 8 elements. Is this visible? You have got sgpa of 8 elements. Now student dot name is a pointer here, it points to the name of this array. Student dot roll number 3 is this element sorry roll number 3 is this element and student dot sgpa 2 is this element. And if I write simply student, it is a pointer to this structure. Now if name is the first element in the structure it may also, these two may be identical.

So for a student, storage allocation will be done corresponding to the total size of this 25 characters, 8 characters 8 floating point and another 8 floating point, alright. And that whole thing will be one block of data called student. The start of that will be a pointer, the starting address of that will be the address of student. And then depending on how they are stored, normally they are stored in consecutive one after another, so you will have this array followed by this, followed by this, followed by this, alright.

So let us see one simple program in which we will read in, we will define this structure and then we will read in the name, the roll number and the sgpa values and then compute the cgpa value and print it, alright. So let us see a simple program which will do that which will possibly make our thoughts and understanding clear. Is it visible? So again include std io dot h main, here we declare i and j we declared as 2 integers, x and y as two floating point numbers, a as a character, these we will use locally.

(Refer Slide Time: 17:30)

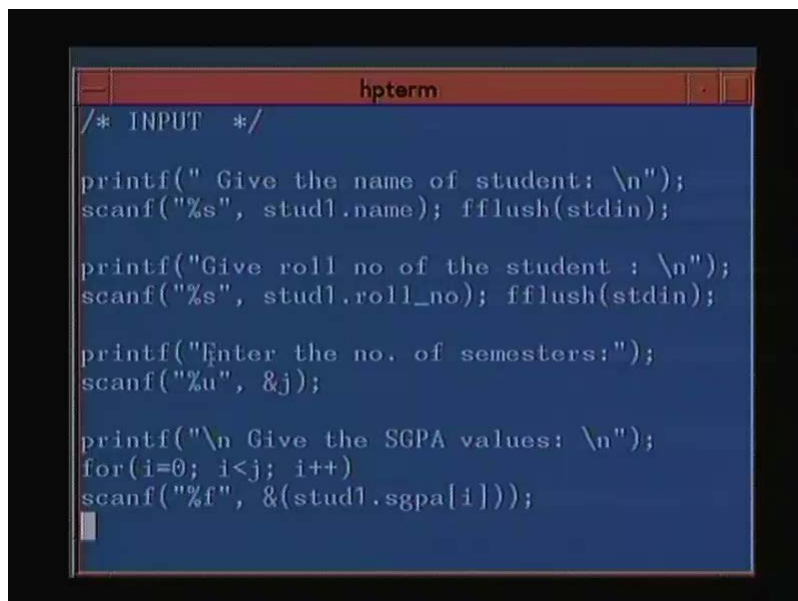


```
hpterm
#include<stdio.h>
main()
{
/* DECLARATIONS */
int i,j;
float x,y;
char a;
struct{
    char name[25];
    char roll_no[8];
    float sgpa[8];
    float cgpa[8];
} stud1;

/* INPUT */
"cgpa.c" 44 lines, 826 characters
```

Here we have defined the structure struct char name 25, char roll number 8, float sgpa 8, float cgpa and the name of the variable is called stud 1. Now we take the input, first is give the name of the student is printed and then in % s format we take stud 1. name.

(Refer Slide Time: 18:55)



```
hpterm
/* INPUT */

printf(" Give the name of student: \n");
scanf("%s", stud1.name); fflush(stdin);

printf("Give roll no of the student : \n");
scanf("%s", stud1.roll_no); fflush(stdin);

printf("Enter the no. of semesters:");
scanf("%u", &j);

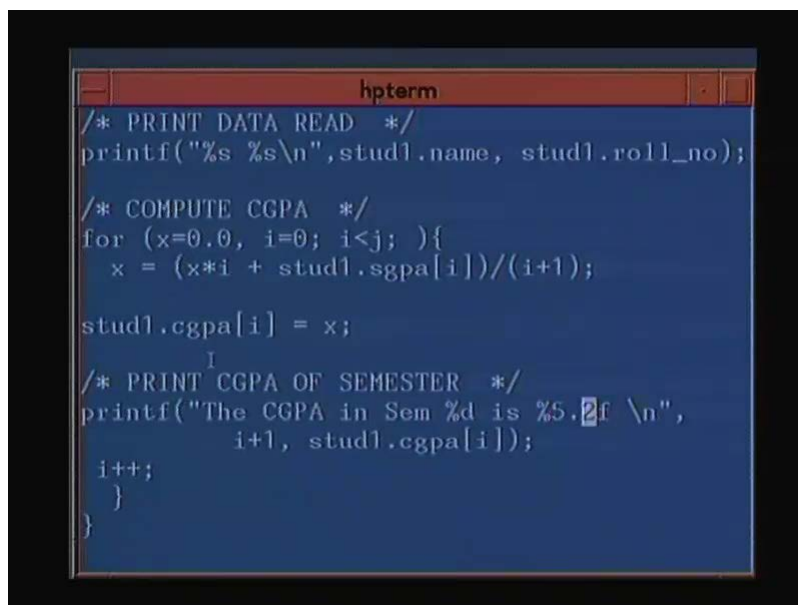
printf("\n Give the SGPA values: \n");
for(i=0; i<j; i++)
scanf("%f", &(stud1.sgpa[i]));
```

So we read the full string of the name in the % s format and stud 1. name is the start address of the array name inside stud 1, fflush std in. Again similarly we take in the roll number, next we take in the number of semesters which is stored in the variable j and number of semester we have taken in just to introduce the unsigned, we saw the unsigned

format previously then in a loop we take in the sgpa values. So this is how we read in the sgpa values, for i equal to 0 as long as i is less than j i plus plus in % f format.

Now, stud1.sgpa [i] is the location of that sgpa value. We have to give the address of that location in order to read in a floating point number so we put the whole thing in brackets and give the address. If we don't put it in brackets there may be lot of other problems depending on the implementation, so it is safe to put in the brackets always. So this is the address of this location because scanf requires address of the location. So in a loop we read in all sgpa's and then we print in the data read in % s format again stud 1. name and stud 1. roll number.

(Refer Slide Time: 21:55)



```
hpterm
/* PRINT DATA READ */
printf("%s %s\n",stud1.name, stud1.roll_no);

/* COMPUTE CGPA */
for (x=0.0, i=0; i<j; ){
    x = (x*i + stud1.sgpa[i])/(i+1);

    stud1.cgpa[i] = x;

    /* PRINT CGPA OF SEMESTER */
    printf("The CGPA in Sem %d is %5.2f \n",
           i+1, stud1.cgpa[i]);

    i++;
}
}
```

And then we compute the cgpa, initialize the cgpa value to 0, initialize i to 0. See this is a comma, so both of these statements up to a semicolon will be done in the initialization. And then in the loop we check i less than j and there is nothing for the loop control. And here inside the loop **I have to**, in this, the loop for starts from here and ends here. And what we do is we do a standard cgpa calculation for each semester wise x which is the current cgpa into the current number of semesters plus the current sgpa of the i'th semester divided by i plus 1, alright. This i and this i should not be confused because this i when it is the first semester, when it is nothing has been done i is 0 and the first element is in stud sgpa 0.

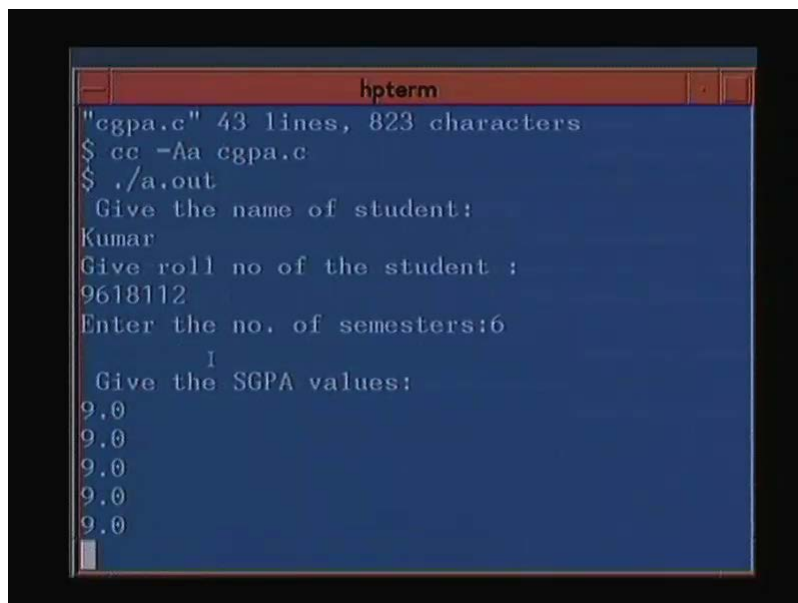
So the fifth semester result is in sgpa 4, that is why this i you compute, multiply all the four semester results by 4 and take the fifth semester result because the fifth semester result is in stud 1. sgpa 4 and you divide it by i plus 1, alright.

So this way you can compute the cgpa of each and every semester and for that semester, you put it in the i'th location x and then you print the cgpa in semester % d is... Now

look at this % 5.2 f, this is a format for... We were normally doing % f. Now 5.2 it will leave 5 places is the total size including decimals, two places after the decimal point. 5 is the total number of places including the decimals and two places after the decimals point. And for a particular semester i plus 1 that is for the first semester cgpa is 0, stud 1. cgpa 0 that is a semester grade for the first semester will be printed and inside the loop you will print the cgpa of each and every semester. We incremented i here and not here, you could have also done it here. I think it is more natural to do it here, looks a bit odd. So is that okay?

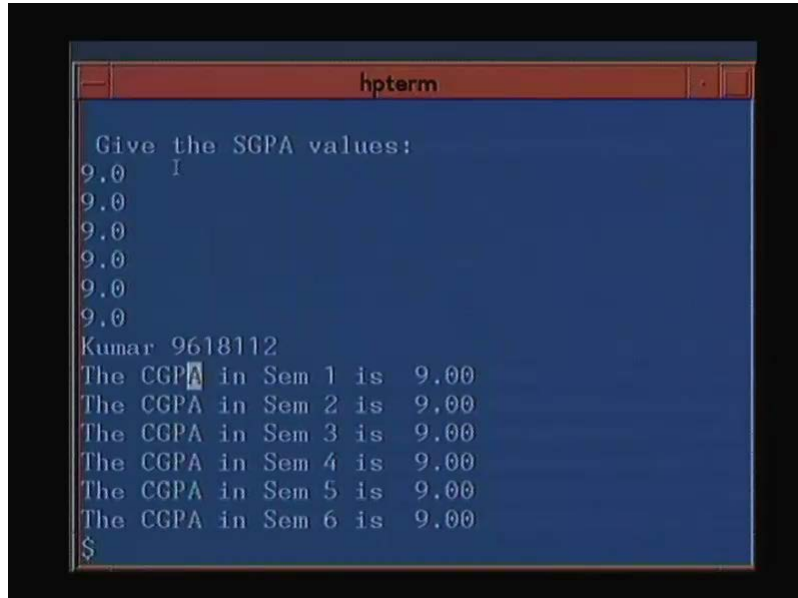
Now let's run it. So what you get printed is kumar, the roll number and I purposely gave all the same just to see that how the print format is coming.

(Refer Slide Time: 23:27)



```
hpterm
"cgpa.c" 43 lines, 823 characters
$ cc -Aa cgpa.c
$ ./a.out
Give the name of student:
Kumar
Give roll no of the student :
9618112
Enter the no. of semesters:6
    1
Give the SGPA values:
9.0
9.0
9.0
9.0
9.0
9.0
```

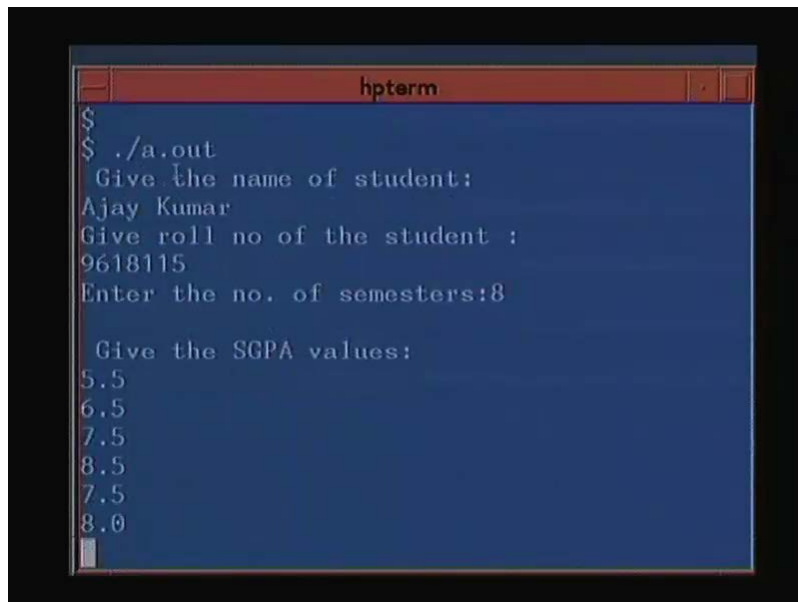

(Refer Slide Time: 23:35)



```
hpterm
Give the SGPA values:
9.0
9.0
9.0
9.0
9.0
9.0
Kumar 9618112
The CGPA in Sem 1 is 9.00
The CGPA in Sem 2 is 9.00
The CGPA in Sem 3 is 9.00
The CGPA in Sem 4 is 9.00
The CGPA in Sem 5 is 9.00
The CGPA in Sem 6 is 9.00
$
```

Now we will give different data to see what will happen if you give this. So though I gave Ajay kumar, it will take only Ajay because it is a string.

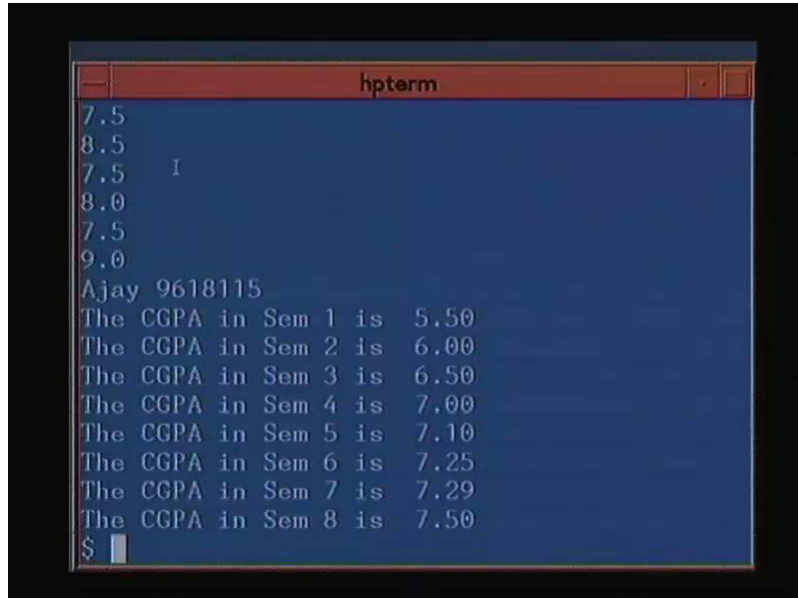
(Refer Slide Time: 24:21)



```
hpterm
$
$ ./a.out
Give the name of student:
Ajay Kumar
Give roll no of the student :
9618115
Enter the no. of semesters:8

Give the SGPA values:
5.5
6.5
7.5
8.5
7.5
8.0
```

(Refer Slide Time: 24:28)

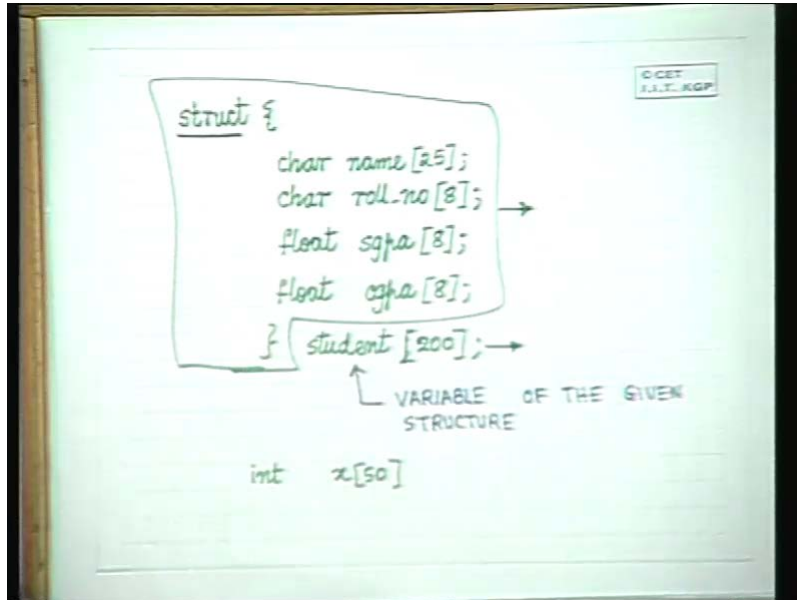


```
hpterm
7.5
8.5
7.5 I
8.0
7.5
9.0
Ajay 9618115
The CGPA in Sem 1 is 5.50
The CGPA in Sem 2 is 6.00
The CGPA in Sem 3 is 6.50
The CGPA in Sem 4 is 7.00
The CGPA in Sem 5 is 7.10
The CGPA in Sem 6 is 7.25
The CGPA in Sem 7 is 7.29
The CGPA in Sem 8 is 7.50
$
```

I would leave it to you to find out how you can take in string with the full including blanks, you can take everything up to the end of line character. There is a way in which you can get the full name. I purposely will leave it to you to find it out. Now, so this is how, see we had first given 5.5 and then we have given 6.5, so this came to 6.0 then we had given 7.5. So, once we gave 7.5, the computation came up here up to 6.5 and similarly we got the semester grade point values up to here.

So you could well realize if you have a program like this that will spoil your first few semesters, you are in trouble. But this is not the only way in which we can define a structure. There are syntactic rules to define a structure. Just before completing that part, suppose we would like to declare an array of students. We could declare this student then this would make it 200, an array of 200 see normally when you write `int x 50`. What do you mean? You mean that x is an array of 50 elements, each element is an integer.

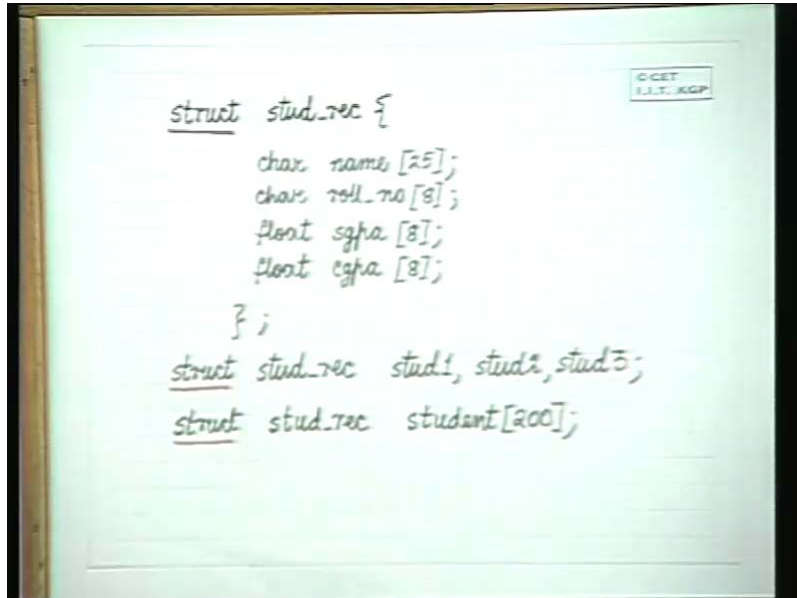
(Refer Slide Time: 27:07)



Student 200, student is a structure of 200 elements, each element is this structure. This is just the type definition instead of int you have given the type definition. Is that understood, this is the type definition. Now you can separately give this type definition and separately give the variable definition. That is here now in C language; we can define the types and then define the variables. Here we have defined the type and the variable together. You can separate out the definition of type and variables and define what are called your own predefined types.

One form, there are many ways in which you can do that also. One form is like this struct stud_rec this is the name of the type, character name, character roll number, float sgpa, float cgpa end.

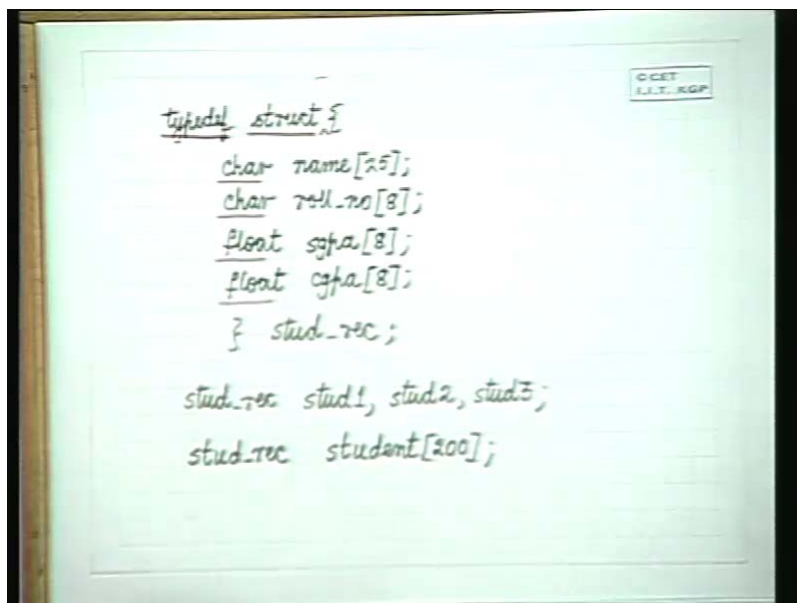
(Refer Slide Time: 28:49)



```
struct stud_rec {  
    char name[25];  
    char roll_no[8];  
    float sgpa[8];  
    float cgpa[8];  
};  
struct stud_rec stud1, stud2, stud3;  
struct stud_rec student[200];
```

And then to define it, you have to give struct stud rec stud 1, stud 2, stud 3 or even stud 3 500. You can again define another one struct stud rec. Look at it this way, stud rec will be replaced by this here, stud rec will be replaced by this declaration here. So you have struct followed by this, followed by this. So this is one way in which you can define a type and do it this way. There is an alternative method to do the same thing. And we will see the advantages of doing this one and the next method and we come to other sort of self-referencing structures later on. This is called the type def way in which you write this is a keyword type def struct.

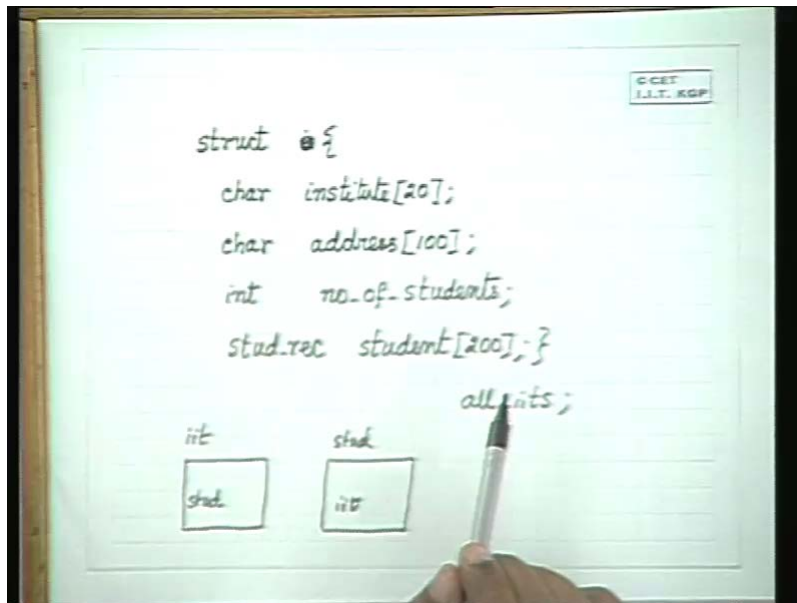
(Refer Slide Time: 30:06)



```
typedef struct {  
    char name[25];  
    char roll_no[8];  
    float sgpa[8];  
    float cgpa[8];  
} stud_rec;  
stud_rec stud1, stud2, stud3;  
stud_rec student[200];
```

So I am defining a type struct, you could also give name here which we have omitted for this case. In some other cases we will see we can also give a name here and why that is important we will come through later on. Character name, character roll number, float sgpa, float cgpa and the name of this type is given at the end here, the name of this type is given at the end here and then you don't have to write struct, you have to just write stud_rect like you write int, char, etcetera you have to just write the type and then declare your variables. You could have also write, you could also define something more, you could use structural definitions to define other structures that is inside a structure, you can have a structure.

(Refer Slide Time: 33:03)

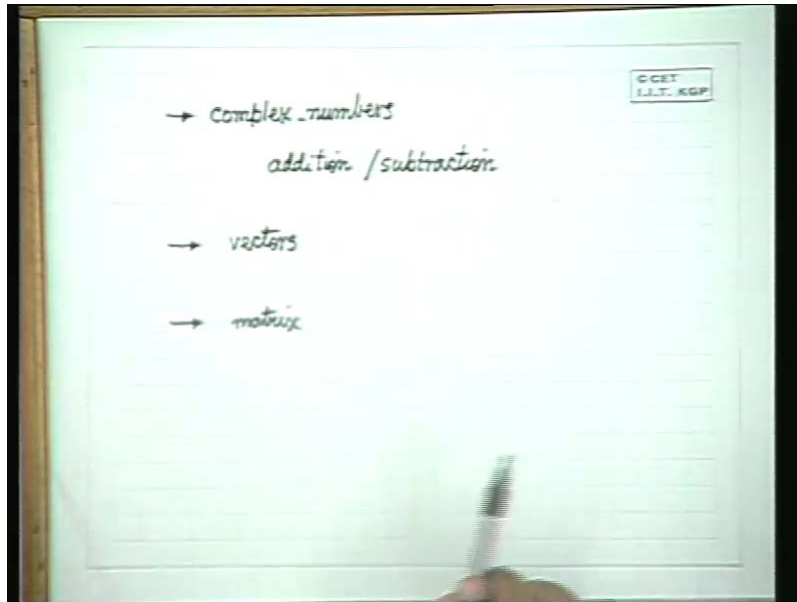


So this is a structure which takes institute, address, number of students and the student record of that institute. So, now you have encapsulated. You have got another structure called all iits where inside this you have got all the student information but you must be very careful that when you define a structure, you cannot have self-referencing in both direction. That is you cannot have normally a structural definition in which here you define this is say IIT, here you define a structure student and in student also you define IIT that is a problem.

You cannot have this because allocation will become impossible. For this it will put in IIT, inside this structure this structure will be present, so the whole thing will become self referencing and you understand the problems of self referencing because you will not know that you end the whole definition. This is for static allocation. When we come to dynamic allocation we will see how we can get it provided, the allocation is not static the allocation is dynamic as and when required. But in static allocation like this, it will not be possible to define structures inside structures in a self-referencing model.

So I would leave it to you to just try out as an exercise, complex, definition of complex numbers and write out functions for addition and subtraction in such a way that you use data structures in a very clean mode.

(Refer Slide Time: 34:50)



Clean mode I mean that when addition is used it will take in two complex numbers and return one complex number, alright. Subtraction will take in two complex numbers as input and return a complex number as output. So now even once we have structures, even a function can return more than one element, it can return an element in a structure. You remember we were discussing that a function has to return a list. How will it return a list? If a function has to return a student record, how will it return a student record?

We can now see, just try out the problem for complex numbers. Another problem that you can try out is vectors. Define a structure for a vector, so you will come to problems of electrical engineering which are more convenient for you may be. Vectors and you can define the dot product the addition, the subtraction, the dot product, the cross product, the cross product of two vectors will return a matrix. So next you can define a data structure for a matrix and then you can write function which will take in a matrix and return determinants and other values like this.

So once we have the use of structures, definition of structures it becomes very convenient to define your own conceptual data structures in C language that makes it very easy to work out a lot of definition which we have in our head and the translation into a programming language is more close to what we have in our conceptual definition rather than converting it into a set of arrays which is just our way of doing it.

So this C language provides you a lot of features for encapsulation of data. So we will study some more of ways in which you can write good programs using such

encapsulation in the next class. We will stop here today and I would request you to try these exercises in your free time.