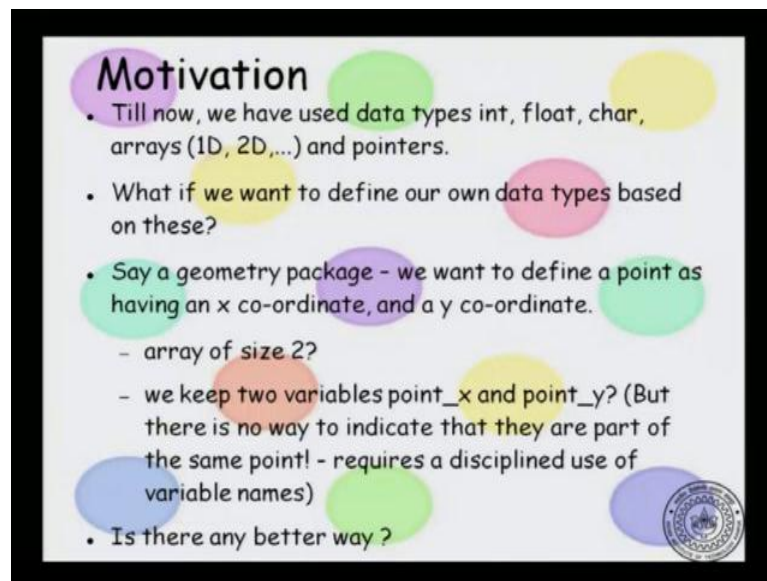


Introduction to Programming in C
Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 49

So, in this lecture will talk about structures and C, which is syntactic feature that C provides in order to define new data types.

(Refer Slide Time: 00:10)



Motivation

- Till now, we have used data types int, float, char, arrays (1D, 2D,...) and pointers.
- What if we want to define our own data types based on these?
- Say a geometry package - we want to define a point as having an x co-ordinate, and a y co-ordinate.
 - array of size 2?
 - we keep two variables point_x and point_y? (But there is no way to indicate that they are part of the same point! - requires a disciplined use of variable names)
- Is there any better way ?

So, let us look at the motivation. Till now we have used the data types that C language has already provided like int, float, character, and also we have seen data types like arrays and we could define arrays of end arrays of character and so on, we have also seen pointers which can hold the address locations of other variables. Now, what if we want to define our own data types using the data types that are already available. So, if you want to define custom data types does the language provide any feature to do it. Before we reach there we will just take a look at why you would want to define such a data type. So, let say that we have designing a geometry package, and we want to define a point on the plane as having a x coordinate and a y coordinate. Now thus a easy way to do it if you have arrays, you could hold a point inside an array of size 2.

Now you would keep the first coordinate in the the x coordinate as the zeroth element in the array, and the y coordinate as the first element in the array, this is one way to do it. Another way is to keep 2 variables point underscore x and point underscore y, and these

are the x coordinates and the y coordinate of a single point, this is another way to do it. But in both these solutions this no way to indicate that these 2 are in intended to be the x coordinate and the y coordinate of the same point. That programmer has to impose considerable discipline in coding in order to maintain this meaning. So, is there a more natural way to do it in C. So, we want to define a point data type, and a point data type internally has 2 integers; one- 2 floating point, one in x coordinate and another a y coordinate.

(Refer Slide Time: 02:18)

Structures

A structure is a collection of variables with a common name. The variables can be of different types (or arrays). Structure variables are called fields.

```
struct point {  
    int x;  
    int y;  
};
```

This defines a structure called point containing two integer variables (fields), called x and y.

```
struct point pt;
```

struct point pt defines a variable pt to be of type struct point.

```
pt.x = 1;  
pt.y = 0;
```

pt

x	1
y	0

memory depiction of pt

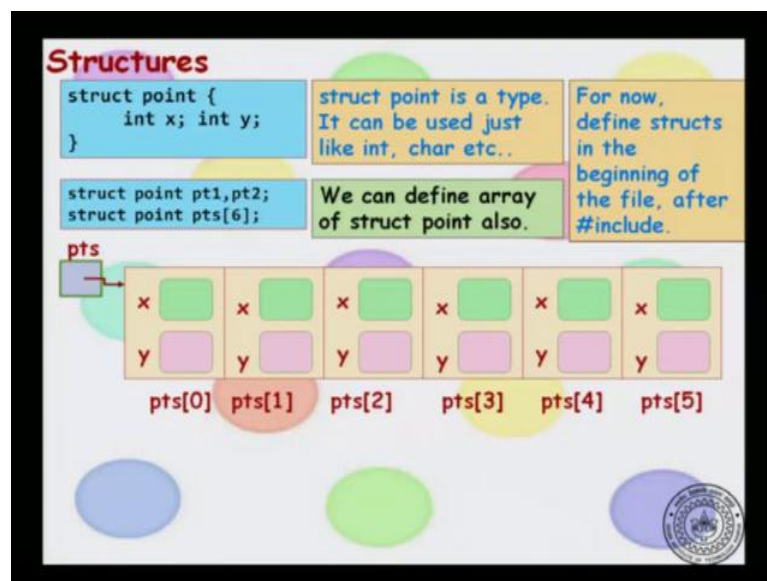
(x, y)

So, now we will define a structure as a collection of variables with a common name. Now the variables can be of different types, this is unlike arrays. We know that in an array you can hold multiple values in a contiguous sequence, but all the values have to be on the same time. So, what is different here with structure is that you can hold multiple values in the same data type, also the same data type can hold multiple sub types within itself. So, structural variables are called fields. So, let us look at a example structure, we define something called a struct point, and it has an int x and a int y coordinate.

And how do you... This is the data type notice that the data type declaration has a semicolon at the end. Now how do you define a variable with this data type, you can say struct point pt. So, pt is 1 variable that is of data type struct point, you cannot say point pt you have to say that it is a struct point. Now how do I assign values to this data type. So, structure is a composite data type that is 2 internal components; one is x, and the other is

y. So, in order to say that the internal components will have certain values I need to say how to you get to these internal components, and this is done by the dot operator. So, you can say `pt dot x equal to 1`, that would assign the `x` field inside the `pt` structure to 1. Similarly `pt dot y equal to 0`, what it will do is it will take the `y` field of `pt` structure and assign it to 0. So, the internal memory representation after executing the statements will be that `pt` is a structure; it has 2 sub fields - `x` and `y`, and `x` will be assign to 1 and `y` will be assign to 0. Struct point is the name is the data type, and `pt` is the name of the variable.

(Refer Slide Time: 04:42)



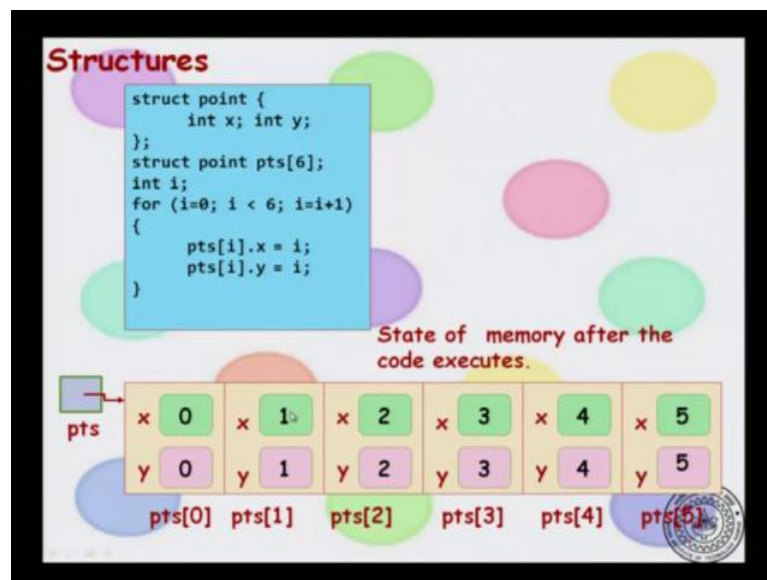
Now as far as C is concerned this structure data type that you define is just like and data type that it provides. So, user defined data types especially structures at ((Refer Time: 04:57)) more or less in the same way as an ordinary data type. And we will see that with an ordinary data type there are multiple things that you can do with it, you can initialize you can declare a variable to be of that data type, we have seen already how to do it? You can initialize a variable of a particular data type, we will see how to do it?

Similarly you can pass a data of a particular type to a function, and return a variable of that type from a function, you will see that all these are possible with structures as well. So, we will see these with examples. Now struct point is just a type and it can be use like any other standard C data type; even though you as a programmer have defined it. For now how do you define a structure, usually you define all the structures that you need at the very top of the file, just of the hash include. Now, you know that with the standard

data type you can define and array of the data type, if you have you can declare array of int, array of characters and so on.

Similarly, you can define a struct point array, C is says that user defined data types are treated in the same way as the standard data types that it provides. So, here let us look at this example. So, we have struct point pt 1, pt 2. So, this say is that pt 1 and pt 2 are two variables which are of type struct point. What about pts 6, it is an array of size 6 where each element in the array is of type struct point. So, to visualize it you would imagine that it is like this; there are 6 cells - each cell contains a struct point. So, each cell will have two fields - x and y. Now how do I assign values to these elements in the array.

(Refer Slide Time: 06:58)



So, you would write a loop, for example of the following form. So, you would say for i equal to 0, i less than 6, i equal to i plus 1 and pts i dot x equal to i. So, pts i is the ith element in the array. In the elements in the array are of type struct point. So, pts i will be a struct point variable, and that variable has 2 fields - x and y. So, I can say pts i dot x equal to something i, and pts i dot y equal to i. So, at the end of execution of this loop the result will be of the following form pts 0 is a structure, and its x and y coordinate are 0; pts 1 is another structure, its x and y coordinates are 1, and so on. So, what are the characteristics features of an array, that it contains cells which are contiguously allocated, so they will be allocated one of to the other in memory, and each cell is of the same type as the others in the same array. So, both those are maintained. These structures

will be stored one after the other in memory also everything in the array is a struct point. So, an array of structs is similar to an array as ints except for the fact that within a cell you will have values that can have subfields.

(Refer slide Time: 08:34)

Functions returning structures

```
struct point {
    int x; int y;
};

struct point make_point(int x, int y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}

int main() {
    int x, y;
    struct point pt;
    scanf("%d%d", &x,&y);
    pt = make_point(x,y);
    return 0;
}
```

- 1. `make_point(x,y)` creates a struct point given coordinates `(x,y)`.
- 1. **Note:** `make_point(x,y)` returns struct point.
- 1. Functions can return structures just like `int`, `char`, `int *`, etc..
- 1. We can also pass struct parameters.

Given int coordinates `x,y`, `make_point(x,y)` creates and returns a struct point with these coordinates.

Now we will see what tells can you do with structures. Just like a variable of a int data type, you can return it from the function. If you declare an int x, you can say return x. Now let say it can be a right functions which can return struct point. So, the theme of this lecture is that user defined structures at treated by c pretty much the same way as the standard data types. So, the behavior should be consistent, you should be able to return a value of a struct of type struct from the function, let see an example. So, here the user defines a struct point, and then when you run the program, you say can give 2 integer values - x and y, and I want to create a point struct with subfields x and y which the user has input. So, I have return a function called makepoint; makepoint takes 2 arguments int x and int y, and what it returns is a variable of type struct point. So, this is the return type of the this whole name is basically the return type of the function, the name of the function is makepoint. Now how do you define it? So, for example, you can define a variable temp of type struct point, and then say temp dot x equal to x temp dot y equal to y, and return temp.

Now, if you forget about this code, particular code, if temp had been an int variable, you would say return temp. If the function was if return type int. Here it is the function is

returning struct point, and you would do it exactly in the same way as a function returning int. So, what else can we do with a normal data type, you could for example pass it as a parameter to a function. So, if you have int variables, you can pass functions taking int arguments. Similarly can you write functions taking struct parameters, and we will see that it can be done, yeah.

(Refer Slide Time: 10:55)

Functions with structures as parameters

```
#include <stdio.h>
#include <math.h>
struct point {
    int x; int y;
};
double norm2( struct point p) {
    return sqrt ( p.x*p.x + p.y*p.y);
}
int main() {
    int x, y;
    struct point pt;
    scanf("%d%d", &x,&y);
    pt = make_point(x,y);
    printf("distance from origin
    is %f ", norm2(p) );
    return 0;
}
```

Using the math library

1. The norm2 or Euclidean norm of point (x,y) is $\sqrt{x^2 + y^2}$
1. To make things easier, we are using the math library whose header file is included by line `#include <math.h>`. The `double sqrt(double)` function is in math library.
2. Use `gcc file.c -lm` to compile file.c with the math library.

norm2(struct point p) returns Euclidean norm of p.

So, we will take an example that is fairly easy to understand. So, you take a point p and calculate the norm of the point p. So, what is the norm of the point p? The norm of the point x y in the Euclidean plane is simply square root of x square plus y square. So, you will just have to calculate a function which does this. For this we will use the math library in C. So, I will say include math dot h, and then I will defined a function norm of struct point p. So, let us call this norm 2 and less ignore why it is call norm 2. So, it is just a function that takes a point p and calculates the norm of p. So, for this what do I do? I will say that returns square root. So, sqrt is the square root function provided by math dot h. So, return square root of p dot x star p dot x plus p dot y star p dot y.

So, the in the main what you would do is user defines user gives an input x and input y. You make a point using the earlier function that we wrote makepoint x y. So, pt will be a point with x coordinate x, and y coordinate y. Now for that point pt you define norm 2 of pt. So, norm 2 of pt here you would pass a point a struct point as a parameter, and the function with calculate the norm of the func of the point and return the norm. So, the way

you would pass a structure is the same as the way you would pass an int or a character or something.

(Refer slide Time: 12:54)

Structures inside structures

```
struct point {
    int x; int y;
};

struct rect {
    struct point leftbot;
    struct point righttop;
};
struct rect r;
```

1. Recall, a structure definition defines a type.
2. Once a type is defined, it can be used in the definition of new types.
3. struct point is used to define struct rect. Each struct rect has two instances of struct point.

rect1 is a variable of type struct rect. It has two struct point structures as fields.

So how do we refer to the x of leftbot point structure of r?

The diagram shows a rectangle labeled 'r' containing two points: 'leftbot' and 'righttop'. Each point is represented by a small box with 'x' and 'y' coordinates.

Now let us take the game a bit further. You know that I can you know that you can defines structures whose subfields are standard C data types. Now, if user defined data types, user defined structures are of the same category as standard data types then I should be able to defines structures whose internal fields or themselves structures. So, we have seen structures whose internal fields can be basic data types, now will see structures whose internal fields are structures themselves. So, let us look at a very reasonable use case in which this can be occurring. So, suppose you want to extend your geometry package, and you want to define a rectangle. Now a rectangle is define by 2 points; 2 diagonally opposite points, let say the left bottom and the right top; these two points define a rectangle So, the left bottom and the right top are themselves points. So, they have some fields which are x coordinate and y coordinate.

So, this is how you would imagine the picture, r is a rectangle - it has 2 points; left bottom right top, and this left bot and right of themselves have 2 subfields x and y, this is the composite picture. Now how do I manipulate this rectangle, how do I say that this rectangles left bottom is the point one, suppose I want to say something like this.

(Refer Slide Time: 14:39)

```
struct point {
    int x;
    int y;
};
struct rect {
    struct point leftbot;
    struct point rightright;
};
int main() {
    rect r;
    r.leftbot.x = 0;
    r.leftbot.y = 0;
    r.rightright.x = 1;
    r.rightright.y = 1;
    return 0;
}
```

The diagram illustrates the memory layout for the `rect` structure. It shows a central box labeled `r` containing two `point` structures: `leftbot` and `rightright`. The `leftbot` structure has `x=0` and `y=0`, while the `rightright` structure has `x=1` and `y=1`. Labels `r.leftbot.x`, `r.leftbot.y`, `r.rightright.x`, and `r.rightright.y` point to the respective fields. A yellow box at the bottom states: "Addressing nested fields unambiguously".

So, again you would use the dot notation, so, is a very consistent representation. So, I would say that in the inside the code of name, I would say `struct rect r` and then I would say `r.leftbot.x = 0`, this says that the take the left bottom subfield of `r`. Now, since that is a structure `r.leftbot` itself has subfields which is `x` and `y`, it is `x` coordinate is assign to 0. Similarly `r.leftbot.y = 0` and so on. So, I will say that this rectangles left bottom is 0, and is 0 0 - and its rightright is 1 1. So, after running this code, this is the state of the memory. So, I will have a rectangle `r`, and its `leftbot.x` is 0, its `leftbot.y` is 0, its `rightright.x` is 1, and its `rightright.y` is 1.

(Refer slide Time: 15:47)

Initializing structures

```
struct point {
    int x; int y;
};
```

1. Initializing structures is very similar to initializing arrays.
2. Enclose the values of all the fields in braces.
3. Values of different fields are separated by commas.

```
struct rect {
    struct point leftbot;
    struct point rightright;
};
struct point p = {0,0};
struct point q = {1,1};
struct rect r = {{0,0}, {1,1}};
```

The diagram shows a coordinate system with a point `P (0,0)` at the origin and a point `(1,1)` at the top-right. A rectangle `r` is drawn with its bottom-left corner at `P` and its top-right corner at `(1,1)`. A point `q` is marked at `(1,1)`.

And now we will also see how to initialize structures. So, we know that normal basic data types like int, char, and all that when you declare a variable, you can also initialize, can you initialize a user defined structure in this way. So, the way we define it is similar to the way you define you initialize arrays. So, initializing structures is very similar to initializing arrays, enclose all the values of the fields in braces, and the values are given in the same order that you they are defined in the structure. Suppose you have struct point int x and int y, how would I initialize it I would say struct point p equal to 0 0. So, this means that the first field int point, that is x is assigned 0. The second field that is y is assigned 0 as well. Similarly if I say struct point q equal to 1 1, it is says q dot x equal to 1 and q dot y equal to 1.

Now, you can do the same thing with nested structures. So, this is very nice. So, if I want to define a rectangle. Remember that a rectangle has two 2 fields, which are themselves structures their points. So, if I say r equal to 0 0 within braces comma 1 1 within braces. What happens is that r first field which is the left bottom, it is will get the value 0 0; that means, that it is the left bottom subfield x will get 0, and the left bottoms y will get 0. Similarly the r is second field is the right top, it will get 1 1. So, right top dot x will be 1, right top dot y will be 1 as well. So, this is how you would initialize an a initialize a structure as very similar to initializing an array. The only thing to remember is that the values must be given in the same order then they are declared in the type declaration

(Refer slide Time: 18:10)

Assigning structure variables

```
int main() {
    rect r,s;
    r.leftbot.x = 0;
    r.leftbot.y = 0;
    r.righttop.x = 1;
    r.righttop.y = 1;
    s=r;
    return 0;
}
```

1. We can assign a structure variable to another structure variable.
2. The statement `s=r;` does this

s	leftbot	righttop
	x	x
	y	y

r	leftbot	righttop
	x	x
	y	y

Before the assignment

Now we know that variables can be assigned to other variables. So, a natural question is can struct variables, we assign to struct variables, and the answer is yes, suppose you have a rectangle `r` whose left bottom is `0 0`, either initialize it you are assign the values, and its right top is `1 1`. So, I define another variable `s` which is also a rectangle, and if I say `s` equal to `r`. Let see what happens? So, at before the assignment `r` is as follows. So, you have `x 0 0 x y 0 0` and `x y 1 1` left bottom on the right top, and `s` is uninitialized it has just been declared, but the no value has been assigned it.

(Refer Slide Time: 19:06)

The slide is titled "Assigning structure variables" in red text. It contains a C code snippet in a light blue box, a list of two points in a yellow box, and a diagram of two structure variables, `s` and `r`, in a light green box. The code snippet is:

```
void main() {
    rect r, s;
    r.leftbot.x = 0;
    r.leftbot.y = 0;
    r.righttop.x = 1;
    r.righttop.y = 1;
    s=r;
}
```

The yellow box contains two points:

1. We can assign a structure variable to another structure variable
2. The statement `s=r;` does this

The diagram shows two structure variables, `s` and `r`, each with a `leftbot` and `righttop` member. The `leftbot` member has `x` and `y` fields, and the `righttop` member has `x` and `y` fields. In the diagram, the values are: `s.leftbot.x = 0`, `s.leftbot.y = 0`, `s.righttop.x = 1`, `s.righttop.y = 1`; and `r.leftbot.x = 0`, `r.leftbot.y = 0`, `r.righttop.x = 1`, `r.righttop.y = 1`. The text "After the assignment" is written below the diagram. A small circular logo is visible in the bottom right corner of the slide.

So, this is the state before the assignment. When you do `s` equal to `r` it is very nice, what it does is `s` is left bottom dot `x` will be assigned 0, `s` is left bottom dot `y` will be assigned 0, `s` is right top dot `x` will be assigned 1, and dot `y` will be assigned 1. So, what happens is goes in to the, it goes in to the structure `r` and copies it entirely in its full depth in to `s`. So, it is not just that a left bottom and right top are copied its internal fields are also copied in to `s`.