

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

Module - 11B

Lecture - 37

More on Structures

Contents

Operations on structures

Functions and structures

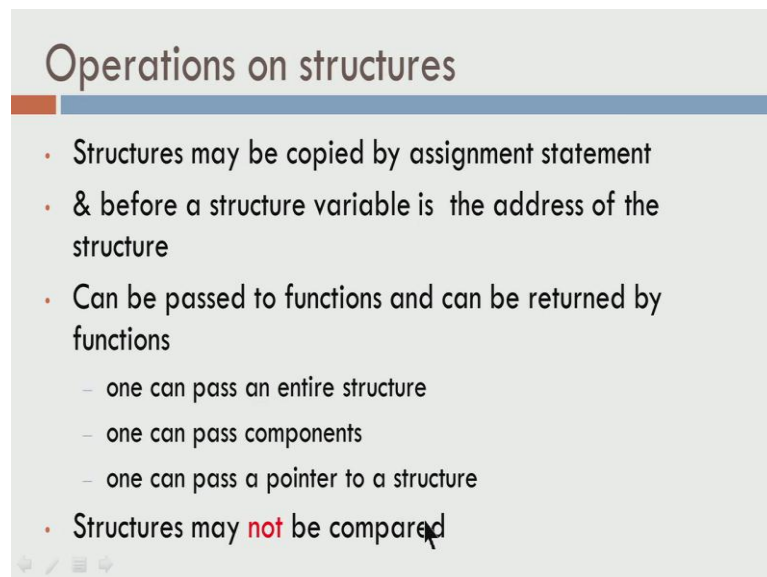
Example: Screen and Centre point

Example: Point inside a rectangle?

Arrays of Structures

Structures and assignment operator

(Refer Slide Time: 00:12)



Operations on structures

- Structures may be copied by assignment statement
- & before a structure variable is the address of the structure
- Can be passed to functions and can be returned by functions
 - one can pass an entire structure
 - one can pass components
 - one can pass a pointer to a structure
- Structures may **not** be compared

There are several things that you can do as Operations on Structures. Structures can be copied by using assignment statement, that is a very common thing, you want to copy, let us say one students record to another student or you want to copy the date to another date and so on, you can do those things. You can put an ampersand before a structure variable and you can get the address of the structure. So, remember structure has in turn has records or members and these members in turn may have other members and so on. All of them get packed as one unit in the memory back to back. If I put ampersand of structure variable, I would get the address of the structure itself. You can take structures and pass them on to functions, we can also make structures return from functions. You

can pass the entire structure, you can pass only the components or you can even pass a pointer to the structure, just like you would do for variables.

For a variable, I could have transferred the variable or I could have passed on a pointer, just like that you could do it here also. The only thing is you cannot compare structures directly with each other. So, if I have int a comma b, I could do if a equals b; however, if I have struct point a comma b, I cannot do is if a equal to b as a check. This is the something that the compiler will catch and give us an error. So, this is probably one major difference between basic data types and user defined data types. You cannot use equality operator or less than, greater than and so on to compare two structures with each other.

(Refer Slide Time: 01:51)

Functions and structures

- Structure as function argument

```
int IsOrigin(pointType pt)
{
    if( pt.x == 0 && pt.y == 0)
        return 1;
    else
        return 0;
}
```

The diagram illustrates the function call and the structure being passed. A curly brace on the right side of the slide groups the function definition and the call 'IsOrigin(myPt);'. Below this, a diagram shows a variable 'myPt' pointing to a structure with 'x' and 'y' fields. Another variable 'pt' is shown pointing to the same structure. The 'x' and 'y' fields are circled in red, indicating they are the values being checked in the function.

Let us see for an example, how structures can be passed on to functions. So, let us say I want to write a small function which is trying to find out, if a point that is passed on to it, is it the origin. So, origin is the point 0 comma 0, I want to find out, if the point that is passed on to it is actually the origin. So, the function is called is origin and instead of passing two integers, I am going to pass the data type called pointType and the local variable name or the formal name is called pt.

So, we have talked about formal names and so on earlier, the formal name is called pt. So, now this is the variable pt, and for pt you have pt dot x and pt dot y, because it is of the type pointType. So, you are checking if pt dot x is 0 and pt dot y is 0, in which case x

and y coordinate are 0 comma 0 respectively, yes it is the origin, I return true else I will return false. So, this is the program here, let us see how this little piece of code would work.

So, let us say I have a point already in place and I call it is origin of my pt. So, let us say this is inside your main function. You are setup my pt to be let us say 5 comma 4, you want to find out is 5 comma 4 origin or not. So, when you make this function call, what happens is remember, we can see that this is actually passing by value, functions and you have something which is passed on by value. So, pt is something which is local and it has two things namely x and y, and my pt is in the caller and caller has two things x and y.

And when you see this function call what happens is, just like what happens for basic variables. The value of the structure is copied to here, what I mean by that is, the value of the x member is copied to the x member of pt, and value of the y member of my pt is copied to y member of pt. So, at this point pt has copies of the x and y members of my pt. Now, you can do this check is pt dot x is 0 and pt dot y is 0, so you are going to go and check these values and you return 1 or 0.

So, the keep thing to notice is that if you make a change at pt, it is not going to reflected my pt, because you made a copy of my pt to pt. The reason being, this is the function call where everything is pass by value, we are not passing the reference to my pt.

(Refer Slide Time: 04:42)

Structures and functions

- Structure as return type

```
pointType MakePoint(int x, int y)
{
    pointType temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

Observe there is no confusion between the two occurrences of x and y

Let us look at another example, I want to make a new point with the given x comma y values. So, given a x comma y value, I give the x coordinate and y coordinate and I want a new data type of the type `pointType`. So, this function we are going to call `MakePoint`, it takes two integers `x` and `y` and it returns a data type `pointType`. So, internally we have a declaration called `pointType temp`. So, this is local to `MakePoint` and you change the `x` member to the integer that you got and the `y` member to the integer `y` that you got.

So, let us see how this sequence would have work, you would have received, let us say 5 comma 10 as the two integers. Now, if you say `pointType temp`, you have `temp` which is a structure and it is supposed to get two members `x` and `y`. Now, we say `temp dot x` equals `x`, so I said `x` was 5, `y` was 10, these are the two values that we have passed. So, `temp dot x` equals `x` would copy the contents of `x` into `temp dot x`. So, `temp dot x` becomes 5 and `temp dot y` equals `y`, copies the content of `y` into `temp dot y`, so this becomes 10.

So, at this point your `temp` has the value 5 for `x` and 10 for `y` and just like any other data type, you can take this and return it, that is what we have in this piece of code. So, again the key thing to notice is that we had copies of integers `x` and `y` and these two integers `x` and `y`, there is no confusion of this `x` and `y` with respect to `temp dot x` and `temp dot y`. `Temp` is a structure and `temp dot x` is a member within the structure, whereas `x` is a free variable, it is of the data type integer. So, `temp dot x` is a qualification to get to the member of this structure called `pointType` and in this case, it is of the particular in the variable `temp`. Whereas, this `x` is a free variable of the type integer, it is not part of the structure.

(Refer Slide Time: 06:58)

```
A screen and its centre point

struct rect screen;
pointType middle;
pointType MakePoint(int, int);

screen.pt1 = MakePoint(0, 0);
screen.pt2 = MakePoint(1920, 1080);
middle = MakePoint(
    (screen.pt1.x + screen.pt2.x)/2,
    (screen.pt1.y + screen.pt2.y)/2
);
```

Declarations

Code using the structure

So, let us see a few other examples, so let us say I have some declarations of this type. So, I want to represent this screen that you see here as a rectangle, so I assume that the rectangle is already declared as a data type before. So, I have struct rect screen and I have done a type def for point, so I do pointType middle. So, middle is going to be a point and I want to find out given a screen, what is the middle point in the screen? So, we want these things.

So, to find out the middle point, so let us say I start with 0 comma 0 which is the left bottom of my screen. When I call MakePoint of 0 comma 0, this is going to return a structure. So, the return data type here is a structure and on the left side, we have screen which is a rectangle, but rectangle dot pt1 is a point. So, on the right side you have a point and the left side you have a point, you have matching data types, everything checks out.

So, we have already mark the rectangles left bottom as 0 comma 0. I want the rectangles right top to be 1920 comma 1080. This is the resolution of my screen, I want the right top to be 1920 comma 1080. So, again when I call MakePoint, this is going to return a point. So, the return data type for this is a point and I want this to be setup as the right top coordinate for screen, which is a rectangle. So, these two lines take care of setting up the rectangle with the appropriate left bottom and the right top points.

And finally, I want to find out what is the middle of the screen. So, maybe it is somewhere here, for my screen, I want to find out what this point is, so to do that I take the bottom coordinate of the screen and the top coordinate of the screen, I add that and divide by 2 and I take the left coordinate of the screen and the right most coordinate of the screen, I add them up and divide by 2 that would give me the center of the screen.

So, these two things are integers and MakePoint takes two integers and returns a data type called point. Left side is already of the data type point and the right side expression gives the data type point. So, both the sides are of the same data type, so if we assume that these declarations are already in place, this code using the structure will given a screen it will tell me, what is the middle point of the screen is.

(Refer Slide Time: 09:40)

Point inside a rectangle?

```
/* IsPtInRect: return 1 if point p is in rectangle r,
   else return 0 */
int IsPtInRect(pointType p, struct rect r)
{
    return (
        p.x >= r.pt1.x && p.x < r.pt2.x
        && p.y >= r.pt1.y && p.y < r.pt2.y
    );
}
```

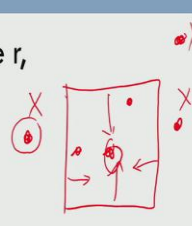


Diagram illustrating a rectangle with a point inside and a point outside. The rectangle is defined by its bottom-left corner (pt1) and top-right corner (pt2). A point p is shown inside the rectangle, and another point is shown outside. A callout box states: "assume pt1 is left bottom coordinate and pt2 is right top".

So, let us say I want to give you a point and I want to find out, is this point inside this screen or not. So, I am going to write a function called IsPtInRect. So, you can think of it as a question I am asking you, is this point inside the rectangle. And I am going to pass the rectangle as r and going to pass the point as p. So, I could ask this about any such point, it could be something outside the screen or inside the screen, I do not know.

I want to find out, if it is outside I want to get false and if it is inside the triangle, I want to get true. So, I have let us say this as my rectangle and I could give you point p as this or point p as this or point p as this and so on, it could be of any of these things. Now,

how do we find out, whether this point is actually within the triangle shown here. So, I want to find out, whether this point is actually within the triangle.

So, to do that if I know up front that the x coordinate of the point is to the right of the beginning of the screen and the x coordinate is to the left of the right side of the screen, then the point is somewhere in between the left and right side of the triangle. And if it also happens to be above the bottom of the triangle and below the top of the triangle, if all these four conditions are satisfied, then the point is within the triangle.

So, this point is not within the triangle, even though it is within the wide range of the triangle, it is not in the x range. So, it is to the left of my left most point, so this point is not a point in the triangle, so or this point, this point and so on, these would not be in the triangle. For all these points, you would get false and points like this I have started with, so point like this and so on will get true.

So, this is a small piece of code which actually takes two different structures, one which is a rectangle and one which is a point and remember rectangle r has pt1 and pt2 has x member and a y member. So, pt1 if it is the left bottom of point, it will have its x and y coordinates and we are comparing that with the coordinates passed on by variable p.

(Refer Slide Time: 12:14)

Arrays of structures

```
struct point{
    int x;
    int y;
} pointArray[10];

pointType pointArray[10];
```

```
struct point {
    int x;
    int y;
} pointArray[] = {
    { 1,2 },
    { 2,3 },
    { 3,4 }
};
```

x ₁	x ₂	x ₃
y ₂	y ₃	y ₁

So, just like basic data types, you can also do a few other interesting things with structures. For instance, let us say I want an array of 10 points, maybe I want to define a polygon which has 10 sides, I could do it using this. So, `struct point {int x, y};` and `pointArray` of 10. So, what this gives us is, we define a structure called `point` with two members `x` and `y`. We do not want just one variable of the type `point`, we want 10 variables of the type `point`.

The variables are going to be called `pointArray` of 0, `pointArray` of 1 and so on, all the way up to `pointArray` of 9. So, you could also do it the way, it is given here, you can just stop with the structure description that is given here on the top and then ask for `pointType pointArray` of 10. So, this is also a valid way of asking for `point`, so in this case I assumed that there is `typedef struct point is given as pointType`.

And finally, even if you have arrays, you can initialize them. So, this is an array of structures on the right side, so `pointArray` is an array of structures. So, you can think of it as I have an array and each one is a structure and the structure has `x` and `y`. So, this is `pointArray` of 0 comma `x` and this is `pointArray` of 0 comma `y`, this is `pointArray` of 1 comma `x` and this is `pointArray` of 1 comma `y`, this is `pointArray` of 2 comma `x`, this is `pointArray` of 2 comma `y` and so on. So, in this case the `pointArray` has three points, so the `x` will get 1, the `y` will get 2 at `pointArray` of 0. Then, `pointArray` of 1 dot `x` is 2, `pointArray` of 1 dot `y` is 3, and finally this would be 3 comma 4. So, this is the set up that we have for array of structures.

(Refer Slide Time: 14:09)

Accessing member values

- Assigning values to structure elements

```
pointArray[0].x = 1;
pointArray[0].y = 2;
```

OR for example

```
pointArray[i].x = 5;
pointArray[i].y = 5;
```
- Printing elements of Structures

```
printf( "%d, %d ", pointArray[0].x, pointArray[0].y);
```

You can access the individual members using arrays as given below. So, you can say pointArray of 0 will be a structure. Given that structure, you can access dot x and get the x member or you could even use pointArray of i to get the ith structure or ith member in the array and in that you want the x coordinate and y coordinate to take the values 5 and 5. You can also do read of these values, pointArray of 0 dot x and pointArray of 0 dot y can be used as they are given here and printed and so on.

(Refer Slide Time: 14:49)

Assigning Structures to Each Other

- Structures can be assigned using the assignment operator

```
struct point newPoint;
newPoint = MakePoint(4,4);
```

So, finally structures can be assigned to each other, we have been seeing this for a while now. So, if I did `MakePoint 4, 4`, `MakePoint` is a function which is going to return a point which it is x as 4 and y as 4. And since the right side is a structure of the data type `point` and I should have the left side also to be of the same data type. In this case, `newPoint` is also of the same data type. So, what it would do is, whatever is return from here, it does member wise copy. It copies each member on the right side to the corresponding members on the left side. So, you can assign structures to each other.

So, this is the useful thing, because you want them to be treated as basic data types. So, this brings us to the end of this module on basic structures, and we will see how we can do a few other things with structures by passing them onto functions and so on in more detail in the next module.