

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

Module - 8A

Lecture – 09

What is an array? Why are they needed?

Example of array: Find average temperature for a year

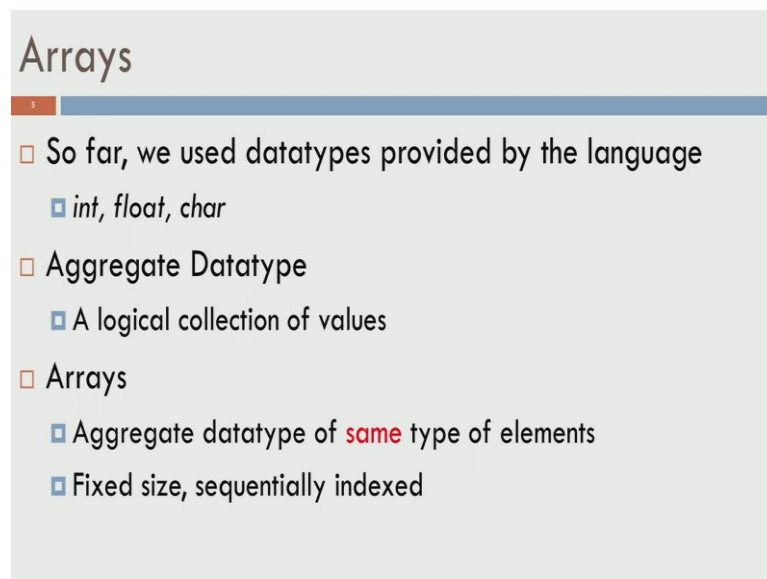
Memory point of view: Array initialization

Example: Find hottest day of a year

Multi-dimensional arrays

Welcome to lecture three. So, in the last two lectures, we saw basics of C programming; and we also saw basic notions of variables and so on. We looked at control structures, we also looked at the notion of loops. So, in this lecture, we will see something really important that C provides us, and that is the notion of arrays.

(Refer Slide Time: 00:34)



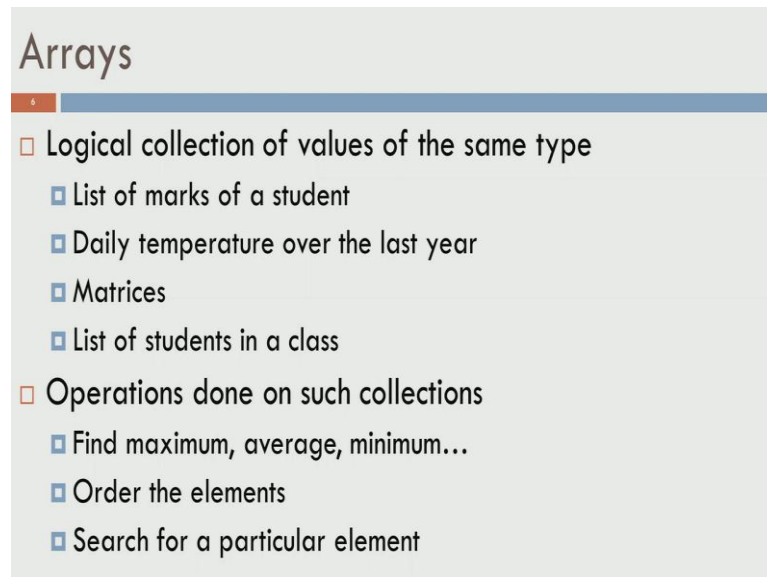
Arrays

- So far, we used datatypes provided by the language
 - ▣ *int, float, char*
- Aggregate Datatype
 - ▣ A logical collection of values
- Arrays
 - ▣ Aggregate datatype of **same** type of elements
 - ▣ Fixed size, sequentially indexed

So, we have seen basic data types like integers, float pointing numbers and characters. And these are basic data types. So, you can get numbers or letters using that. But sometimes we also need a logical collection of these values; it is not that, just one value is sufficient; we need a logical collection of values. And this is where arrays come in. So, arrays are what are called aggregate data types. So, it is an aggregate data type of a specific ((Refer Time: 01:03). It aggregates data of the same type of elements. You can also aggregate data of different types together; we will see that later. These are called

structures. We will see them later. But arrays are aggregate elements of the same data type. So, there are lots of examples, where you need this. One primary thing is these arrays are usually fixed size and they are also sequentially indexed. So, we will see all these things as we go along.

(Refer Slide Time: 01:31)



The slide is titled "Arrays" and contains a bulleted list of topics. The first main bullet is "Logical collection of values of the same type", which includes sub-bullets: "List of marks of a student", "Daily temperature over the last year", "Matrices", and "List of students in a class". The second main bullet is "Operations done on such collections", which includes sub-bullets: "Find maximum, average, minimum...", "Order the elements", and "Search for a particular element".

So, arrays are logical collections of the same type. For instance, it could be list of marks of a student, it could be temperature that you are recording over a year or you want to save matrices and do operations on matrices, and so on. So, these are all examples of arrays. And on these arrays, you want to do operations like find minimum, find maximum or you want to order all of the elements in such a way that, the names of students come in alphabetical order or you may want to search for a particular temperature of the day and so on. So, these are common operations. You want to be able to read entries into an array; you want to do operations on an array, and so on. And that is exactly what we will do in this lecture.

(Refer Slide Time: 02:14)

Imagine this: Find Average Temperature of the Year

Code Segment:

```
float temp1, temp2, ..., temp365;
float sum, average;

scanf("%f",&temp1);
scanf("%f",&temp2);
.
.
.
scanf("%f",&temp365);

sum = temp1 + temp2 + ... + temp365;
average = sum/365;
```

So, let us do a little thought experiment. I want to find out average temperature of the year. So, I have 365 days. And if I did not have support for these aggregate data types called arrays, my program would look something like this. So, I show only the code segment here. Let us see what it has. So, there is first of all 365 days; I will need 365 variables; float, temperature 1, temperature 2, and so on up till temperature 365. I will need all these declarations. So, even though I showed dot dot dot, you really need so many declarations. And then I am going to find out the sum and the average – find the sum and therefore, find the average. So, one thing that I have to do is take all the variables one at a time and read through them. So, I have so many declarations and then I go on and scan all of them.

And finally, I need a huge expression, which takes each of these variables and add them up and give sum. So, what we have is something really cumbersome. So, we have 365 variable names; we will have to scan each one of them; and we will have 365 lines in which we will scan them. So, that is what you will do in this block here. And finally, this one, where you are adding up all the values is also rather cumbersome, because if you forget something, you have no way to go back and check which one is missed out. So, this is not a nice way to do things. And programming languages give support for the same data type; temperature being floating point and have many of those aggregated into something called an array. So, let us see the same thing done using arrays.

(Refer Slide Time: 03:59)

More Elegant: Arrays (Example 1)

Code Segment:

```
float temp[365];
float sum=0.0, average;
int i;

for(i=0; i<365; i++){
    scanf("%f",&temp[i]);
}

for(i=0; i<365; i++){
    sum += temp[i];
}

average = sum/365;
```

365 elements of the same type

Scan the elements one by one and store

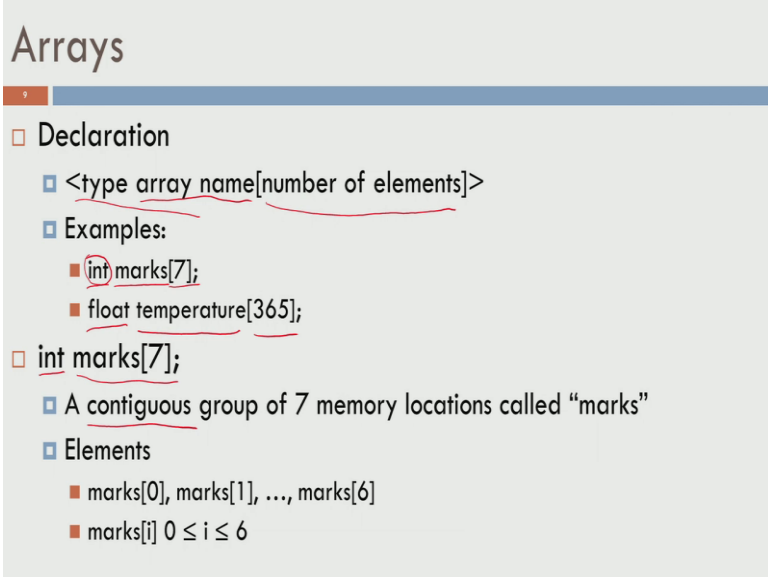
Add the elements

It is much more elegant if we use arrays. So, the first thing you will notice is that, there is float temp of 365. So, what you see here is you see that, there is a declaration here; which goes like open bracket 365 close bracket. All it says is I do not want a single floating point number; I need 365 floating point numbers. And... So, that has to be allocated somewhere. As before, we have float sum equals 0 and average. And now, I want to scan the elements one-by-one. This whole thing becomes very nice and simple; you have seen the for loop before. For i equals 0; i less than 365 i plus plus; scanf ampersand temp of i. So, what this loop does is it scans the elements one-by-one and it stores it in what is called temp of i. We will see what temp of i means in a little while. But, the first thing I want you to do observe is that, let us become much more elegant. Instead of 365 lines that you had earlier, now, you have three lines of code, which is scanning 365 elements.

The other thing that we have very nice is just the addition itself. So, as before, we iterate over all the elements from i equal to 0 to i equals 364 – both inclusive; and we add the temperature to sum. So, you add one temperature at a time and the result goes back to sum. At the end of this loop, you have the sum, which is the sum of all the temperatures over 365 days. So, if you divide by 365, you get average. So, one thing that you have to notice here is that, the sum is actually initialized to 0 here. Therefore, you start with 0 as the temperature and then you add on temperature of each of the days to give temperatures of 365 days; and then you divide by 365. So, the nice thing about this

whole thing is it fits into one screen. So, if I want 5 years, all I have to do is ensure that, I iterate over 365 into 5 iterations. So, that piece of code is still going to be something very very small. So, I do not have to declare so many variables; and the program does not get clumsy; and there is less scope for errors when you have arrays than when you have individual variables.

(Refer Slide Time: 06:28)



Arrays

- Declaration
 - `<type array name[number of elements]>`
 - Examples:
 - `int marks[7];`
 - `float temperature[365];`
 - `int marks[7];`
 - A contiguous group of 7 memory locations called "marks"
 - Elements
 - `marks[0], marks[1], ..., marks[6]`
 - `marks[i] 0 ≤ i ≤ 6`

So, let us go and look at arrays in more detail now. The first thing is – as before, we need to declare arrays before we use them. So, the syntax is as follows. So, you start with type and then you give an array name and then you give number of elements. So, just like this – you specify the type; you specify the array name; and you want the number of elements. So, in this case, marks is an array of size 7 and they are all integers. So, the key thing is – remember – this is an aggregate data type of type integer here. Similarly, in this line, we have temperature, which is an array of 365 elements; and the values that it can contain are floating point values. So, one important thing that you have to remember is that, when you say int marks of 7 for example, you get something, which is contiguous in nature; as in, these are locations, which are continuous in your memory. And the individual elements of the array can be indexed as marks of 0, marks of 1, and so on up till marks of 6. So, anything which is of the form marks of i, where 0 less than or equal to i, less than or equal to 6 is valid.

(Refer Slide Time: 07:50)

Memory point of view

```
int marks[7];
```

- Each element can be thought of as a variable
- Just like individual variables, they start out uninitialized
- Values can be assigned to elements
 - ▣ `marks[3] = 36;`
- '&' is used to get the location in the memory
 - ▣ Just like it was for a variable
 - ▣ `&marks[1]` would be 2732

Address: 2731	-	marks[0]
	-	marks[1]
	-	marks[2]
	36	marks[3]
	-	marks[4]
	-	marks[5]
Address: 2737	-	marks[6]

Let us see what happens from the memory point of view. So, earlier, we saw what happens to variables from the memory point of view. Let us see what happens to arrays from a memory point of view. So, when you see a declaration like `int marks of 7`, what you are really seeing is an array. So, you see memory locations here; I am showing only a segment of the memory. And instead of one variable called `marks`, we are going to have 7 variables named `marks of 0` to `marks of 6`. So, this is not a mistake; it is `marks of 0` to `marks of 6`, and not `1` to `7`. So, some languages start indexing arrays at `1`; but C indices start at `0`. So, you can think of it as 7 variables namely, `marks of 0` to `marks of 6`.

And as we did earlier, if you do not have any initialization, the values in the array are unknown. You should assume that, these values are unknown. And this could get laid out anywhere in the memory just like variables do. The only thing is that, these are going to be 7 contiguous locations. So, for instance, let us assume that, `marks of 0` was allocated address `2731`; then `marks of 1` would be at address `2732` and so on; `marks of 6` would be at address `2737`. So, this is what I meant by arrays are going to have contiguous set of locations.

So, as I said, each element can be thought of as a variable. Just like individual variables, they start out to uninitialized. One nice thing is once you have declared an array; just like in variables, you can assign values to these variables. So, we saw that, we can have left-hand side variable name, right-hand side value or expressions for variables. The same

thing applies for arrays. So, in the left-hand side, you have an individual variable – marks of 3; and on the right-hand side, we have a value 36. So, if you do this in your program, then the value will go to 36. So, at this point, you know the contents of the location marks of 3. As before, we can use ampersand to get the location of the memory. So, for instance, if I want let us say address of marks of 2, then I do ampersand of marks of 2; and ampersand of marks of 2 would give me 2733 as the value; ampersand of marks of 1 would give me 2732 as the value, and so on. So, the addresses are going from 2731 down to 2737 in increasing values. So, they are contiguous and they are increasing from 0 to array index 6.

(Refer Slide Time: 10:31)

Revisit the Example

Code Segment:

```
float temp[365];  
float sum=0.0, average=0.0;  
int i;  
  
for(i=0; i<365; i++){  
    scanf("%f",&temp[i]);  
}  
  
for(i=0; i<365; i++){  
    sum += temp[i];  
}  
average = sum/365;
```

The diagram shows a code segment with three annotations in pink boxes:

- A pink box with the text "365 elements of the type float" has a pink arrow pointing to the `temp[365]` in the first line of code.
- A pink box with the text "Read into memory location of temp[i]" has a pink arrow pointing to the `&temp[i]` in the `scanf` function call within the first loop.
- A pink box with the text "Loop runs from i=0 to i=364, a total of 365 times" has a pink arrow pointing to the `i<365` condition in the second loop.

So, let us revisit the example that we did earlier. So, we have float temp of 365; say we have an array of type float and 365 values. At this point, we do not care about where it is getting laid out in the memory, because that is something that, your run time system should do. So, as a programmer, we do not care about where it is in the memory location; just like for variables, we do not care where they are in the memory. Then this is something that, I did not draw attention to earlier. Now, let us see what it is doing. If it is an individual variable, we saw that scanf takes the format and the address of a variable; the same thing is happening here. We have the format here and we have the address of a variable; only that, the variable is not an individual variable; it is actually an array location. So, temp of i is temp of 0, temp of 1, temp of 2, so on till temp of 364. And when finally, when you look at this loop, it runs from 0 to 364; so it actually runs 365

times. So, your array is indexed from 0 to 364; the loop also runs from 0 to 364. We are actually scanning 365 elements. And in this loop, we are adding all the 365 elements into sum and you average at the end of it. So, it is as simple as that. So, from arrays, we get contiguous locations and we have a same data type; but we have many elements of the same data type one after the other.

(Refer Slide Time: 12:07)

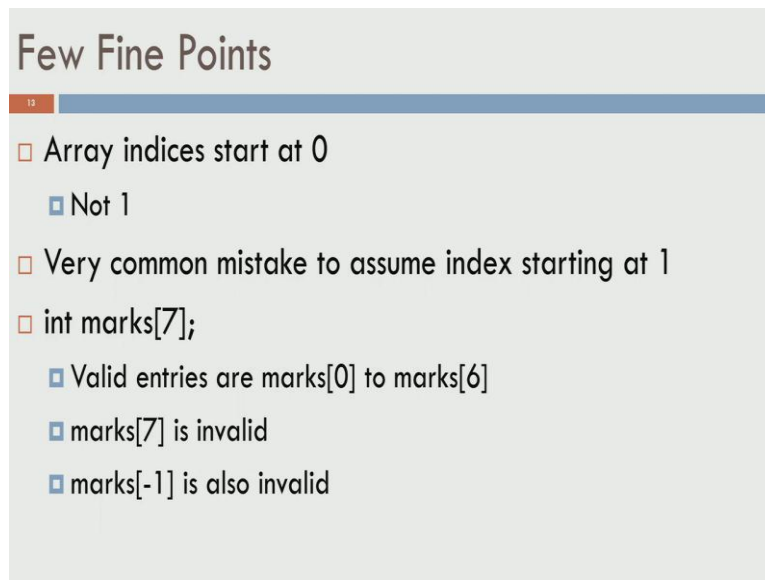
Initialization

- Arrays can be declared with initialization
 - ▣ `int marks[7] = {22,15,75,56,10,33,45};`
- New values can be assigned to elements
 - ▣ `marks[3] = 36;`

22 ✓	0
15 ✓	1
75	2
36	3
10	4
33	5
45 ✓	6

There are various ways to initialize arrays; you can do something like this. Let us say I want marks of 7 students and I could have it as int marks of 7 and read it from the user. Or, sometimes I know what these values are. So, I could initialize it right away; I could do this. So, int marks of 7. So, I am declaring marks to be an integer array of 7 elements. And you can specify the list of values on the right-hand side using a curly braces. So, in this case, marks of 0 would be 22; marks of 2 would be 75; marks of 6 would be 45 and so on. So, the locations on the left side can be seen as 0 to 7; and the values can be seen from left to right. So, you get a one-to-one mapping from left side to the right side. So, when you do this, let us say these are locations 0 to 6 and the values are 22 for 0, 15 for 1, and so on till 45 for marks of 6. It is not just that you can initialize and leave it at that; you can change the values just like you do for other variables. So, you can do something of this effect. Even though you have initialized, mark of 3 to 56, we can do some assignment like this – marks of 3 equal to 36; and that will change the value to 36. So, we saw that, the value changed from 56 to 36.

(Refer Slide Time: 13:35)



The slide is titled "Few Fine Points" and contains a list of items:

- Array indices start at 0
 - ▣ Not 1
- Very common mistake to assume index starting at 1
- `int marks[7];`
 - ▣ Valid entries are `marks[0]` to `marks[6]`
 - ▣ `marks[7]` is invalid
 - ▣ `marks[-1]` is also invalid

So, there are a few fine points that you have to remember. Array indices always start at 0 in C. If you come from other programming languages, some languages start at 1. So, be aware of this. This is a common mistake and lot of people get trapped in this thing that, the indices start at 1 and not at 0. So, in C, they do start at 0. And when you do marks of 7, marks of 0 to 6 – all are valid; marks of 7 is invalid as well as marks of minus 1 or minus 2 and so on. So, you will never have indices, which are negative; nor, you will have an index, which is greater than and equal to the declaration that you had. So, for example, in the float that you showed earlier, so we have 365. So, temp of 365 would be invalid, because we have only temp of 0 to temp of 364.

(Refer Slide Time: 14:35)

Example 2: Find the Hottest Day

Code Segment:

```
float temp[365], max;
int hottestDay, i;
/* NOT SHOWN : Code to read in temperature for the 365 days*/
hottestDay = 0;
max = temp[hottestDay];
for(i=1; i<365; i++){
    if (max < temp[i]){
        hottestDay = i;
        max = temp[i];
    }
}
printf("The hottest day was Day %d with temperature %f", hottestDay, max);
```

Assume Day 0 is the hottest and record the temperature.

If the i^{th} day is hotter than our current record, update

So, I want to talk about another small example, which is along the lines of finding the hottest day. So, I have read 365 temperatures into an array called temp; but I want to find out the maximum temperature or which day was the hottest day. So, what I want the user to print is print the day in which the day was the hottest as well as a temperature of that day. So, as before, we have float temp of 365; and I have two arrays – two variables namely, hottest day and i. I am going to use hottest day to find out which day is the hottest. And I am going to use i to iterate over all the locations. So, initially, what I am going to assume is day 0 is the hottest. So, let us say this is January 1; I assume that, January 1 is the hottest and it is day 0. So, temp of hottest day, which is temp of 0 goes to max.

Now, I run a loop from 1 to 364. So, this goes from January 2 till December 31. And what I am going to do is I am going to see if the current temperature is less than the... The current maximum temperature that I have in record is less than temperature of day i. So, if it is less, then max is that I have is not the actual max; some other day became hotter. So, I have to update it. So, I have hottest day equal to i updated. I not only update the day in which the temperature was very high; I also record the actual temperature, because that is what you are actually tracking. So, what this loop really does is this. So, you go from day 1 to day 364. So, we start at day 0; we go from day 1 to day 364. And if some other day becomes hotter, we update it and we move forward.

And since I have to find out the hottest day in the year, I have to go all the way till 364 days or day 0 to day 364 – 365 days. At the end of it, hottest day will have the index of the day and max will contain the actual temperature. So, I can print that, the hottest day was day number hottest day with temperature max. So, this is a very simple loop. I did not show code for reading in temperature of the 365 days, but we only see the code for finding out the maximum. So, this is the very simple and common example of how arrays are used. I want to find out the maximum. So, it could be marks of 100 students or so. And I want to find out which student got the highest mark. So, I may have to just iterate over this and find out the largest value. So, we start with day 0 being the hottest. And if i-th day is hotter than the current record, update it.

(Refer Slide Time: 17:45)

Multidimensional Arrays

- Arrays with two or more dimensions can be defined
 - ▣ Useful for matrices, 3D graphics etc.

`int A[4][3];`

	0	1	2
0			
1			
2			
3			

`float B[2][4][3];`

	0	1	2	0	1	2
0						
1						
2						
3						

There are also multidimensional arrays that are possible; so not everything in practical uses 1-dimensional. So, even though I have... So, I have things like matrices and so on. Naturally, they are multidimensional or in this, matrices are actually 2-dimensional; you could also have 3-dimensional structures and so on. So, C gives you flexibility to have arrays of multiple dimensions. So, for instance, let us see something of this kind. If I have a declaration `int A` square bracket 4 square bracket 3; it means that, we have four rows numbered 0, 1, 2, 3. So, that comes from this. And number of columns being 3 numbered from 0 to 2. So, we have four rows by three columns.

I can also have something, which is 3-dimensional. So, in this case, I have a floating point array called B. And there are three dimensions to it. So, you can think of it as x, y and z dimension. So, in the x dimension, there is 2; you can think of it as two planes: plane 0, and plane 1. In each plane, I have four rows and three columns. That is what you see here. So, in each plane, I have four rows and three columns. So, this comes in very handy for handling matrices and graphics and so on. We will look at these examples in a little later. In a little while, we will see examples of multidimensional arrays. So, at the end of this module, what we have is basic notion of arrays; how we can use the arrays, how we can index them, and what happens inside memory. So, in the next two modules, we will see more details about arrays.