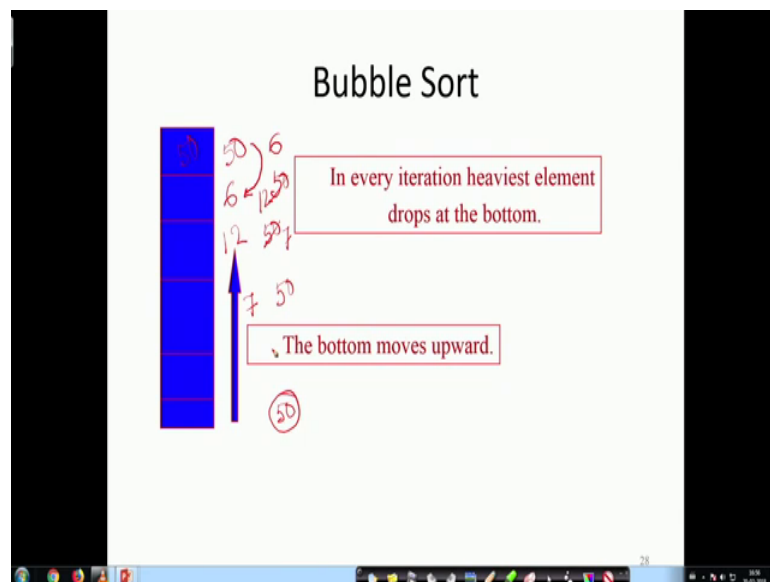


Problem Solving through Programming in C
Prof. Anupam Basu
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture-46
Bubble Sort (Contd.)

So, we were discussing about the Bubble Sort Algorithm and the approach as was discussed in the last class was that every element, every iteration pushes the heavier element at the bottom and that is what we had and the lighter elements therefore, goes up.

(Refer Slide Time: 00:33)



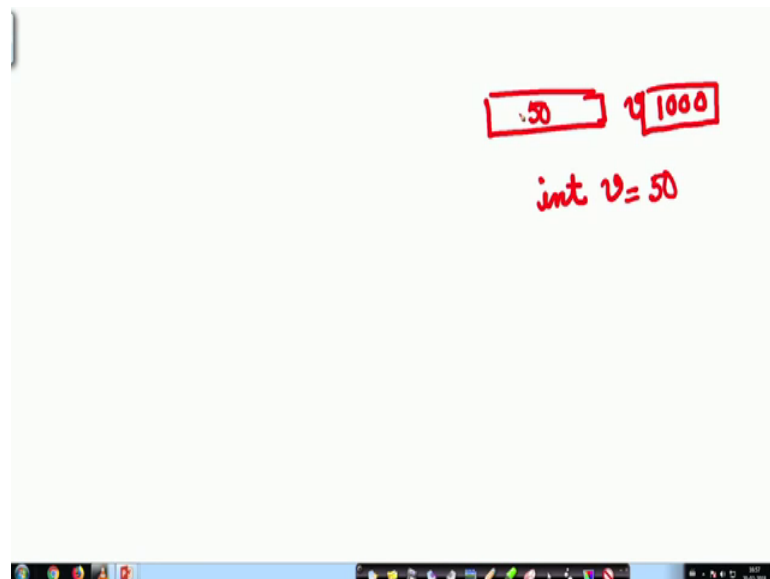
So, we had shown an example a couple of examples by which we can show that the heavier elements go down and the other lighter elements go up for ex we had given a number of examples to that effect.

So, if there be 50 here I do not know this visible let me write it here 50 here and 6 here and say 12 here and 7 here. And what will happen is 50 will be swapped with 6 because 50 is heavier than 6 so, larger than 6. So, first it will be 6 and 50 and then 50 will be compared with 12 again there will be a swap. So, there will be 12 50, 50 will be compared with 7. So, it will be 7 50 in that way ultimately the 50 if it is heaviest will come at the bottom.

So, we swap only if the element at the top is heavier there by heaviest element is coming to the top and the others which are less than this, I mean less than this heaviest element will be going at the top bottom moves upward. Now, we are going to write the algorithm for this or the program, for this before writing the program we need to understand a very interesting and fundamental concept of C programming that is pointers.

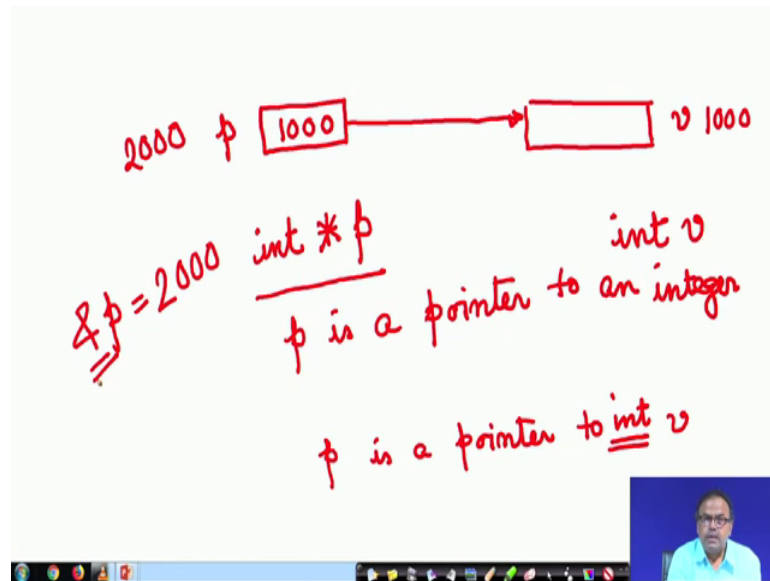
We will not formally introduce pointers here, but we will be talking about pointers in a different way in a context of this bubble sort. Now, what is pointer? Pointer is something that points right now suppose there is a variable here suppose the variable is named v and till now we have seen that we can write declare the variable as `int v` ok.

(Refer Slide Time: 02:51)



Now, suppose I do not so, v has got a particular address may be that address is 1000, 1000 is the address of v. Now, if instead so, if I say `int v equals 50`; that means, in the address 1000 50 is written.

(Refer Slide Time: 03:43)



Now, suppose we have got this address v and which is an integer.

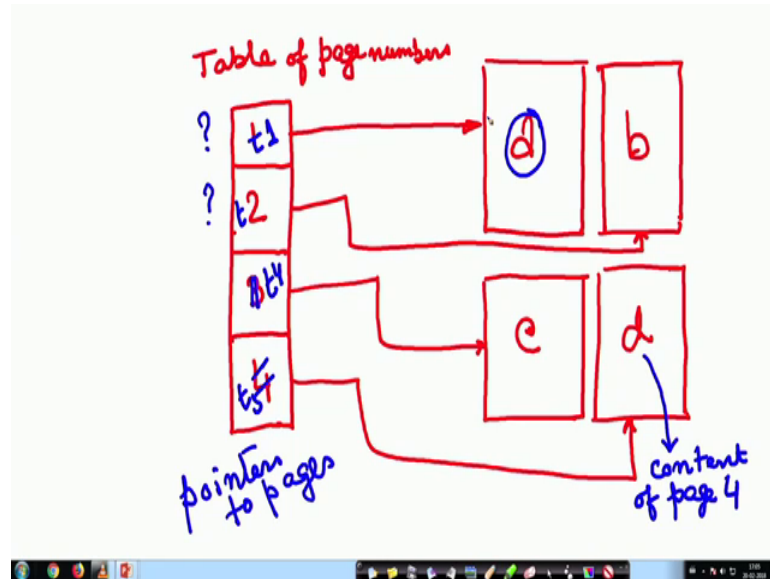
But I am not declaring v as such, but there is another variable p, which will always point to some integer. So, p will point to this v right p is pointing this v. In that case I can say p is a pointer to v now. So, therefore, what did I say I said that v is stored in the location 1000, now pointer p this variable p therefore, stores the value 1000 ok.

Now this v is of type integer. So, v is of type integer. So, v is a pointer to an integer v that we know because v I have earlier said that v is an integer, but say so, I can I can very well say int v, but what is the type of p p is a pointer to v, which is an integer. So, we write that as int star p, what does it mean? It means that p is an is a pointer to an integer, if I write it I have it is not concerned with v only. It says that p is a pointer to an integer ok.

So, what is the value of p now value of p is 1000, we need to say and what is the address of p the address of p now this where is this p this is in location 1000 and where is this might be in location 2000. Therefore, what is the address of p and of p is 2000 we know that this and operator gives us the address, we have encountered this during scan f when we are discussing about scan f right.

So, let us have this diagram to understand the concept of pointer suppose there are different things written in different pages of the book all right.

(Refer Slide Time: 07:08)



This 1 page, this is another page this is another page and this is another page each of these pages have got a page number. Now what I have I have got a table where suppose this has got some content a, this has got some other story b, this has got some other story c, this has got some other story d.

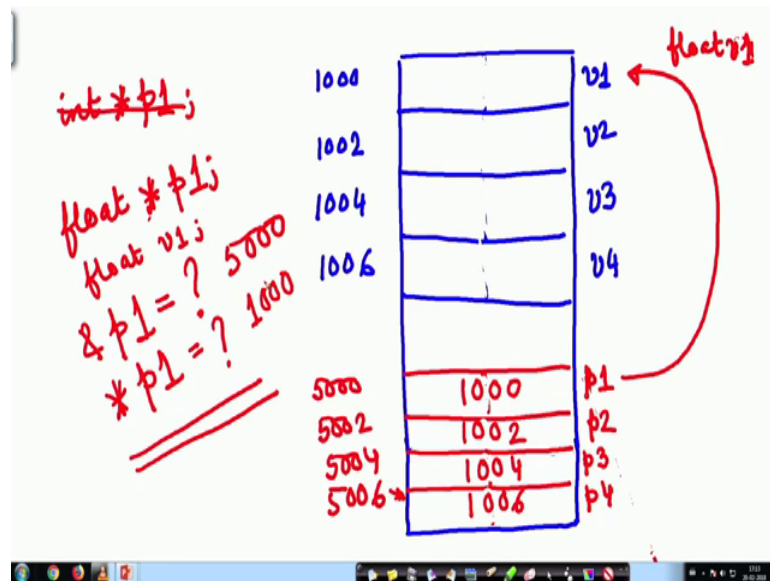
Therefore, the content of each of this pages are the stories a, b, c, or d, but which page I am looking at suppose here is a page number table of page numbers. So, I have got a 4 pages. So, the pages are 1 2 3 4. Now, suppose I want to I just want to come to this page number. So, this page number does not tell me what story is there, but this page number tell me that I have to go to this page this entry is telling me this particular page number is telling me that, I have to go to this page this is telling me that I have to go to this page and this is telling me that I have to go to this page.

So, what are these what are these what are these things, what are these, what are these are nothing, but pointers to pages and what is a a is a content of the page. So, a b c or d say is content of page 4 and these are pointers to the pages; that means, if I want to go to page 4 where should I go, because this page may be located at some other table may be suppose this pages of a book are torn out and have been kept on different tables ok.

So, page number 1 is giving me the table address. So, that it is on this particular table. So, it can be on table t 1, this can be on table t 2, may be the third page is on table. So, third page is on table t 4 and this is on table t 5 it is possible. So, the first table I have to

if I want to read the first page I have to go I will come here the first entry this is the point in to table t 1. So, and I go to table t 1 and to find this particular page and read the content in that all right. So, these are pointers to pages, but these pointers also have to be in some memory location let me draw the diagram in different way, now suppose this is my memory and I have got 4 variables.

(Refer Slide Time: 10:57)



Suppose I have got 4 integer variables and each integer variable takes 2 bytes am showing 2 bytes here 1 after other all right, this is the variable v 1, this is the variable v 2, this is the variable v 3, and this is the variable v 4 right.

And similarly let me just change the color. So, these are suppose in locations sorry let this be in locations v 1 is in location 1000 assuming that an integer takes 2 bytes this one is in location 1002, this one is in location 1004 1002 1004 and this is in location 1006.

And let me extend my table bit, now suppose I have got here again 4 integers this is say this is sorry things have mixed up a little bit I want to come to this. So, so here I have got p 1 is a pointer variable with again, since it is a integer let me say p 2 p 3 and p 4, if their integer pointers have not bothering about that am not bothering about how many locations it is taking.

Now this p 1 is actually I am now not bothering about this division here 2 bytes am just showing those 2 bytes together that will be easier for me to write all right. So, here the p

1 is storing the value 1000 why p 1 is a pointer to this variable. So, this p 1 tells me not the value of v 1, but it tells me where v 1 is located.

So, 1000 is a position where v 1 is located. So, p 1 is storing that pointer and p 2 is say 1002 p 3 is 1004 and p 4 is 1006. Now these locations also have got address suppose these addresses are 5000, 5002, 5004 and 5006 this 5006 ok.

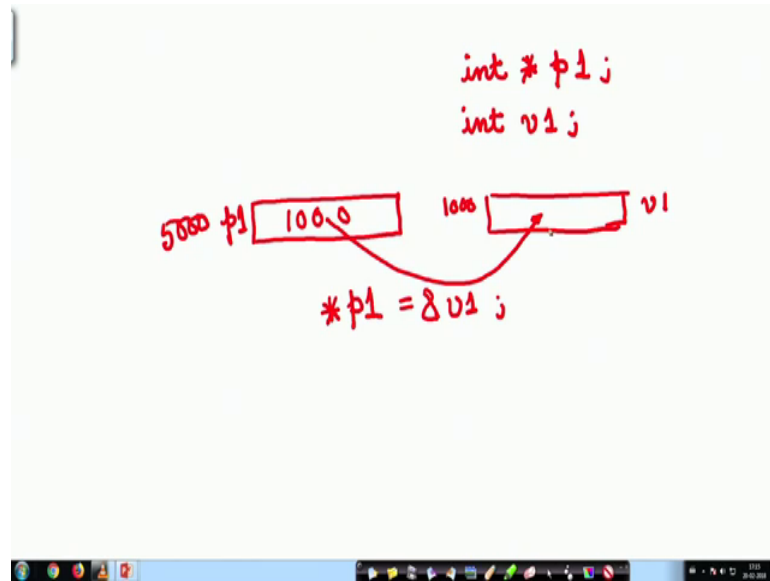
So, please try to understand when I say p 1 what is p 1 p 1 is a pointer to an integer ok. So, `int *p 1;` that means, that p 1 is an integer all right and p 1 is an integer sorry p 1 is a pointer integer variable ok. It could be a floating point variable. So, had it been a floating point number. Then suppose is v 1 v 2 v 3 v 4 am just for the sake of am just for the sake of clarity am just not bothered about, how many locations variable is taking am just erasing this ok, whatever it requires depending on the machine that is taken.

Now suppose each of them is a float. So, float v 1. So, v 1 is a floating point right v 1 is a floating point number. So, then and if p 1 points to v 1, then I should write `float *p 1;` that means, p 1 is pointing to a floating point number ok.

So, since it is a floating point number if. So, if I say now and p 1 what will I get what is the address of p 1 if I try to get this is not an assignment what is and p 1, what is the address of p 1 address of p 1 is 5000 all right and what is `*p 1`? What is the meaning of this please do not be confused. This star and this star has got two different meanings, this star means it is just saying that p 1 is a pointer to pointed to what type of variable pointer to a floating point type of variable.

So, for v 1 am just writing `float v 1` so; that means, that v 1 is purely a floating point number not a pointer, but when I say `float *p 1`, then p 1 is a pointer to some floating number. Now, if I just what is `*p 1` in this case what is a content of p 1, content of p 1, content of p 1 is 1000 all right content of p 1 is 1000. So, let me just repeat it once again. So, we have seen in declaration something like `int *p 1 int v 1` all right.

(Refer Slide Time: 18:47)



Now; that means, p 1 is a pointer p 1 is a pointer, suppose located at location 5000 and p 1 is a pointer to a variable v 1, which is also an integer v 1 located at location thousand all right.

This two lines are just meaning these two things this two pictures this 5000 or 1000 this part is not known to us, because that is known by the compiler not by us, but logically it was this is the picture leave aside this 5000 and 1000 only these 2 parts are declared by this. Now if I say star p 1; that means, the content of p 1, what is the content of sorry yeah content of p 1, whatever is the content of p 1 now if I make p 1 to point to v 1 in that case if I make p 1 point to v 1.

So, what will that be content of p 1 should be assigned to and of v 1 and of v 1 will give me the address of this and the content of p 1 will get that therefore, this 1000 will be written over here; that means, this pointer is pointed to this v 1 ok. I will not go in to further details of this right now, we will come to this as an when the context comes later. Right now we this in mind let us go back to the algorithm of bubble sort, what is being done on the algorithm of bubble sort.

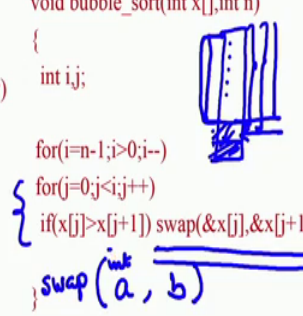
(Refer Slide Time: 20:59)

Bubble Sort

```
#include <stdio.h>

void swap(int *x,int *y)
{
    int tmp=*x;
    *x=*y;
    *y=tmp;
}

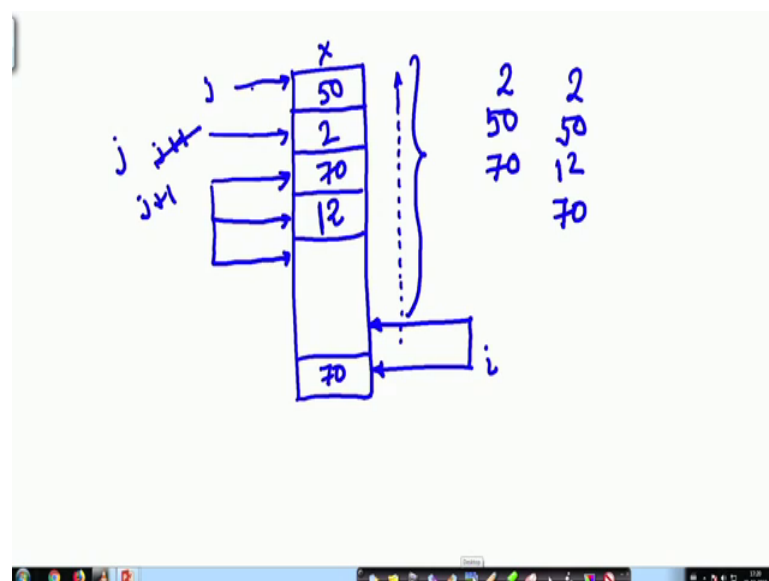
void bubble_sort(int x[],int n)
{
    int i,j;
    for(i=n-1;i>0;i--)
    {
        for(j=0;j<i;j++)
        {
            if(x[j]>x[j+1]) swap(&x[j],&x[j+1]);
        }
    }
}
```



The diagram shows a vertical array of elements. The top element is 50, the second is 2, the third is 70, and the fourth is 12. A dashed line indicates a swap between the second and third elements. To the right, the values 2, 50, 70, 12, 70 are listed. Below the array, a box labeled 'i' points to the bottom element (70). A box labeled 'j' points to the second element (2). A box labeled 'j+1' points to the third element (70). A handwritten note says 'swap(a, b)' with 'int' written above it.

Let us try to understand each of the functions first thing is we have got this standard equation find no issue. Now, first I will start with this function void bubble sort bubble sort is void, because it is not returning any value any time any value if this just sorting some element what array x array and the size of the array is n int ij there are 2 local variable because they are using used as indexes of these 2 loop, in this loop what is happening for I assigned n minus 1 2 0 i minus minus; that means, 1; that means, if I have an array here this is my array I have got I pointing here.

(Refer Slide Time: 22:07)



Because, that is the size of the array minus 1 so, i is pointing here and it will go up to 0. So, gradually it will go in this direction up that is what the first for loop is saying ok. First for loop is saying that i is starting from the bottom and going up up to 0 as long as it is greater than 0 and then for j equal to 0 j minus j equal to 0 to j , less than, i j plus plus if x_j is greater than x_{j+1} ; that means, what if x_j is greater than x_{j+1} .

So, let us again go back to this j is starting from 0 and I am comparing this value 50 with the value 2 with $j+1$ where is $j+1$ if x_j is greater than x_{j+1} , what would I do I would swap. Let us go into here if x_j is greater than x_{j+1} am swapping x_j and x_{j+1} , I will come to the swap function later am assuming that the swap is working working correctly that means it is swapping them.

So, what is happening? So, that is loop will go on j equal to 0 then they will be 1 and in that way it will go on. So, let us see again. So, next when I swap it becomes 2 and 50 right. Now, I have got other values now j is being incremented. So, now, this is becoming j and this is becoming $j+1$ in that way this swapping will be done suppose this is 70. So, there will be no swap no swap.

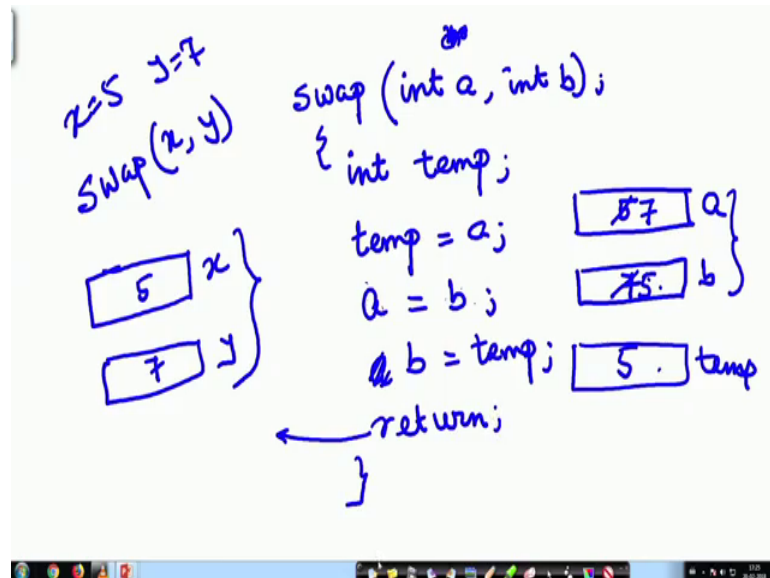
So, let us see here, if this condition does not hold swap is not taking place am going back to this loop all right so, sorry. So, now, 70 remains here and then suppose the next one. So, now, j changes and $j+1$ is pointing here now there will be a swap here. So, this is 12 say 12 and 70 will be swapped. So, 250 12 70 in this way I will go on up to this point and next.

So, ultimately in this iteration ultimately I will have this heaviest element say 70 is the heaviest element coming over here, then i will be decremented and I will come here and then from this zone again I will do the same thing as I did till now right. Exactly that is being done here. So, I will be doing this and then I will increment I will decrement a little bit and I will go on doing this. I will be then looping doing the same loop again for 1 level less I mean the last element is already having the heaviest element. So, I will do it among the remaining part.

So, this part this part is already covered. So, I have decremented it. So, I will be now playing in this zone. And once that is done this part will also be covered I will be playing in this zone I will be playing in this zone and in every zone. So, once every time am deciding on the zone ok, $n-1$ $n-2$ like that it is going on. And every time

am doing this loop for each of the pair wise element comparison and they are by am pushing the heaviest element at the bottom. That is the code for bubble sort how it works this swap keeps a I have not saying this swap. Earlier we have seen that if we do swap in any arbitrary way.

(Refer Slide Time: 27:48)



For example, if I want to swap 2 variables say I write swap int a int b let me do it here let me do it here if I do it like this int sorry if I just do swap int a, int b and inside I use another variable temp, temp gets a b gets a, sorry b gets a, and a gets sorry a is stored here a gets b and then temp has been remembered there. So, b gets temp and so, this was my function return.

So, here suppose a was when it came it was called with x and y s. So, main program called swap with x and y and x was 5, y was 7 right. So, x was a location here 5 and y was a location 7. Now when it came here a was a local location and I know that the functions are called by value. So, 5 was copied in a and 7 was copied in b, 7 was copied in b and then by this temp became 5, b became 5 and a became where is 7 b was stored in a sorry this was stored in this these became 7, and this again became 5 all right that is how it was done.

So, 7 a was 5 so, temp was 5 7 went over here to a and here 5 was copied again here this was fine. So, this swapping was done by a by b and then when I returned no change is reflected in x and y that was the problem.

So, we will now see how the swap function can be implemented, but again another point is I have just done return I and a function can return only 1 value, I could not return a and b to the main function right. So, how can the thing be managed. So, that after the swapping the result is reflected back in the main function we will see that in the next lecture.