Introduction to Problem Solving, and Programming
Prof. Deepak Gupta

Department of Computer Science Engineering

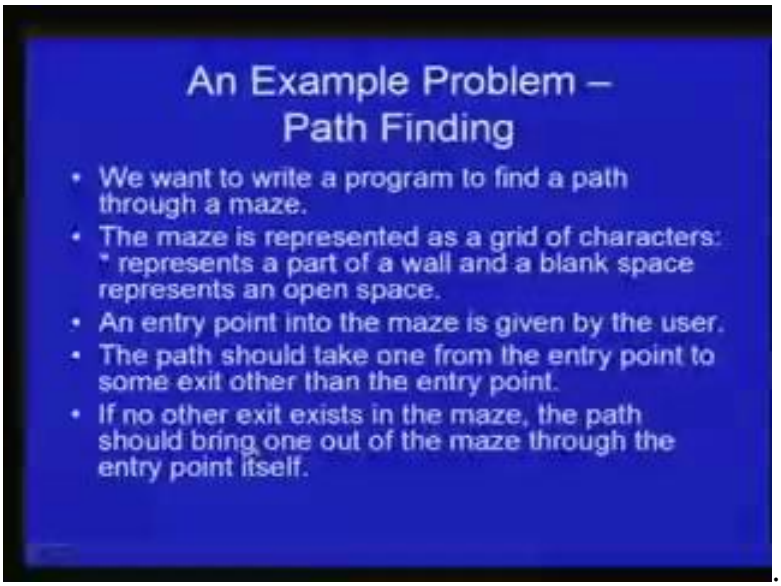
Indian Institute of Technology, Kanpur

Lecture – 16

In the last lecture, we had talked more about functions, and we had introduced notion of scope, and life time, and we had talked about global variables etcetera. And finally, have left I had left with you a problem to think about, and that problem was to write a program to find a path through the maze. I hope you spent some time thinking about the problem. So, let us just review the problem once again we want to write a program to find a path through the maze the maze is represented as a grid of characters.

(Refer Slide Time: 00:51)

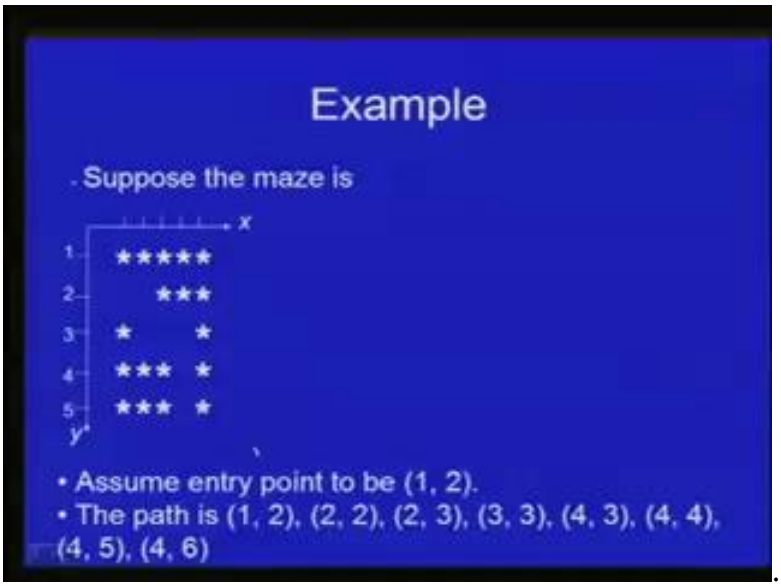
.



.

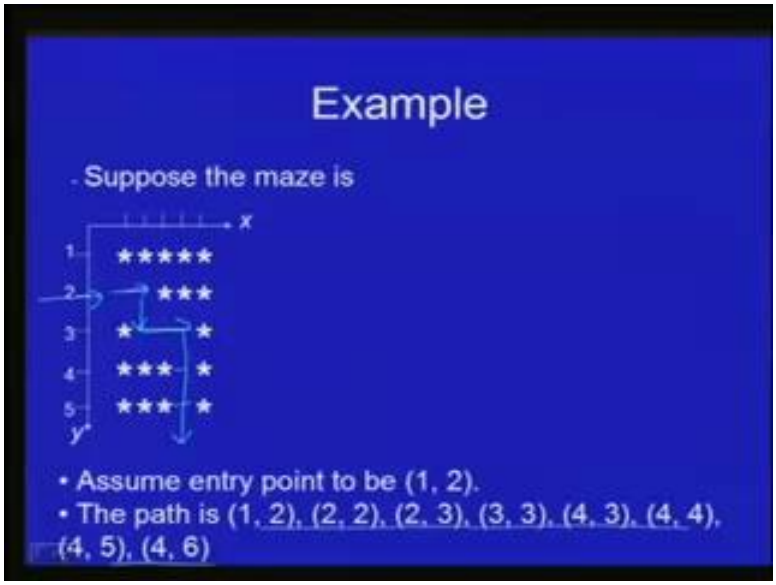
A two dimensional grid of characters a star in the grid represents a part of wall where one cannot go, and a blank space represents an open space, where movement is possible. And an entry point into the maze is given by the user as an x comma y value, and remember that we have said that if there is any other exit in the maze, then the path should take one from the entry point to some other exit in the maze. And if there is no other exit in the maze, and the path should bring one back outside the maze through the same entry point, and this remember was our representation of the maze on the x y plane.

(Refer Slide Time: 01:27)



The star represent walls, and these blanks represent open spaces, and assuming this is the entry point this is the path that the program should tell us to take.

(Refer Slide Time: 01:51)



So, in terms of the coordinate system, this path turns out to be this. Now, this problem might look to be complicated, but as it turns out it has a very simple algorithm to solve this problem.

(Refer Slide Time: 01:54)

I will give you the algorithm, but, and I will leave you to convince yourself that this algorithm works in all cases. So, the algorithm is very, very simple. So, we start at the entry point, and as long as we are not outside the maze that is as long as we are somewhere within the maze we do the following. So, this entire thing is one step in the path.

So, we have to maintain a current sense of direction. That direction could be north south east or west the first thing you do at any place in the maze you turn right, and then if you are facing a wall, then turn left. If you are still facing a wall turn left again if you are still facing a wall turn left again and so on. So, as long as you are facing a wall keep turning left.

Note that you cannot turn left more than four times because, then you'll be facing in the original direction, and if you were if you had come to a particular point from some from some direction, then there is at least one way to move from the current point, which is to go back in the direction from, which we had came we had come,so when when this loop terminates, then step forward in the current direction. So, essentially while we are turning the direction is changing. Now, the reason this algorithm really works is that by trying this we are trying all possible four directions, and seeing which way can go, but we are trying this four directions in a in a certain order. So, if we reach a dead end, and

then this loop will make sure that we turn back, and come back to some place which we had visited earlier.

And then later on we will not go back to a path that we had already visited once that is, because when we come back to the same place. We will be facing in the reverse direction as compared to the direction in which we were facing when we visited this particular place the first time. So, the this is of course, very intuitive explanation of why the algorithm works I would like to encourage you to take some example, and see for yourself that the algorithm in deed works.

And nice thing about the algorithm is that if there is no path in the maze from the entry point to some other exit point, then it will plot a path which brings you back to the original entry point. So, the algorithm is straight forward enough now, we have to focus our attention on how to implement this algorithm as a program, and the first thing that we have to worry about for implementing this as a program is how we are going to represent a maze.

(Refer Slide Time: 04:37)

So, the obvious answer to that is that we represent the maze as a two dimensional array of character right, because it is two dimensional grid, and each character in the array at any I j location would either be a star character which represents a wall or a blank character which represent an open space right. So, maze x y suppose maze is the name of the array what represent the maze character at position x y. Now, since we will also

finally, go outside the maze, and at other places we will check whether we are facing wall or not.

Now, when we are checking whether we are facing a wall or not, we are going to be at we may be at the edge of the maze, and the direction we are facing is outside the maze. So, we have to check that the character facing us is a wall or an open space. We have to make sure that we do not violate array bounds remember that it's very important that when we access an array element the indices index or the index that we specify for the array array element are within the bounds of the array.

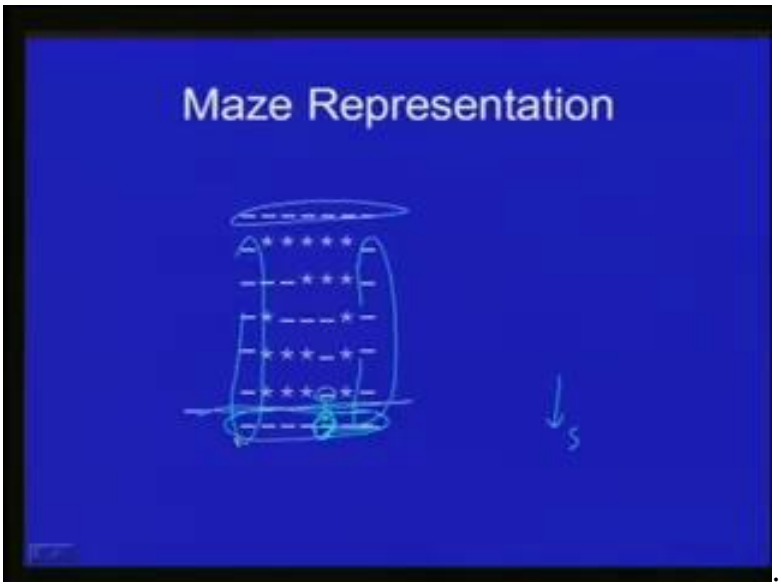
So, we have to make sure that even when we are outside the maze or when we are on the edge of the maze, and when we are checking when we are facing outside the maze. And we are checking, when we are facing a wall or not we have to ensure that we never violate the array bounds, and to make sure that we never do that what we are going to do is to create an open path all around the maze.

So, let's say the user uses a maze of size 5 by 5 what we are going to do is to convert that maze into a maze of size 7 by 7, and in the centre of this size of the maze the original maze given by the user is placed, and all around it on all four direction. There is an open path. So, let me show you what I mean. So, let us say this is the maze that the user has given us I have instead of putting blank characters I have put the dash characters just to show you that these characters exist.

(Refer Slide Time: 06:56)

Now, around this maze, we are going to put a one character thick open path. So, this path we have created outside the maze on all four directions. So, the idea is that even if from this point we come out here, and we are at this location the x, and y positions are still within the bounds of the array.

(Refer Slide Time: 08:00)



And similarly for example, if you are here, and we are facing towards the south assuming that downwards is south, and we want to check whether we are facing a wall or not, then we will have to check this character in the maze. Now, if the maze is if the array is only till here, then this index would be outside the bounds of the array, and will have will have to put special check to ensure that we do not violate the array bounds by putting an extra path all around the maze.

We avoid those kinds of complications. So, the net result is that the size that we will declare for the maze will be two more than the actual maximum size that we want to permit. Let us now try to develop a program for this, but let us first look at a design for this problem this is reasonably complicated problem. So, here is a possible what is known as functional decomposition of the problem that is we can identify this distinct, and independent sub problem.

(Refer Slide Time: 08:04)

So, what we are going to do is to define functions for each of these, and then put together these functions in, and use functions in the main program to solve the overall problem. So, the first function is read maze, which will read the maze from the user the second function will be read starting position which will read the starting or the entry point of the user. Note that we need to make these into function, because even to the complexity even though the functionality of this function might appear simple it is actually a little bit more complex, because you want to make sure that the input given by the user is always consistent, and it's not invalid, and so on so forth.

So, as you will see we need to make a large number of check on the user supplied inputs, and that is why we prefer to um make these into independent function. The next function is outside maze, which will just tell us whether we are currently outside maze or within the maze, and you can see that this function would essentially check whether our current position is on that border around the maze. That is somewhere on the border of the maze that we had created if it is on the border, then we are outside the actual maze otherwise we are inside it.

The next function is facing wall which will tell us whether we are currently in the current direction what we are facing is a wall or not. So, if we are facing a wall we cannot move, and then functions turn left, and turn right, which will change our direction appropriately

the step forward function will move us one step ahead in the current direction, and that will change the current position that we are at in the maze.

And finally, the function print position will use to simply print one position in the path. So, at each step in the path will print the current position. So, we need to write all these functions, and as you can see that most of these functions are going to use number of large amount of common information. So, all of those we will define as global variables.

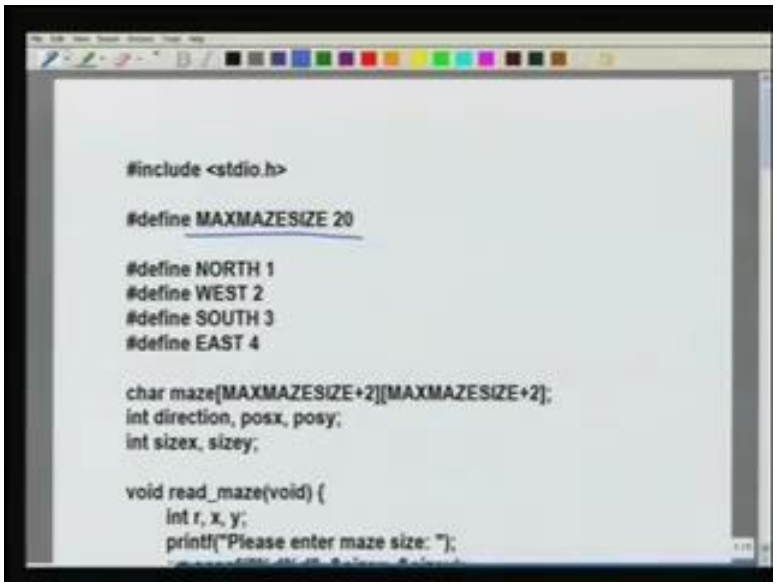
(Refer Slide Time: 10:37)

So, these are the global variables we are going to need the maze itself which is going to be as beside a two dimensional array, then the actual size of the maze as supposed from the as supposed to the maximum size of the maze that we can accept. So, in the sizex will be the actual size of the maze in the x direction, and size y will be the actual size of the maze in the y direction.

And posx posy is the current position within the maze at what point we are within the maze at any given step, and finally, direction is the direction that we are facing in at at the current movement, which could be north south east or west. So, let's now, put together these ideas in the form of a program, and using a bottom up approach lets develop these functions one by one, and then finally, put these together into in an overall program.

So, let's now, look at the program for solving this maze problem. So, we start with some constant global variable declaration. So, this max maze size is a constant fixes what is the maximum size of the maze that the user can give. So, this is in the x direction as well as the y direction.

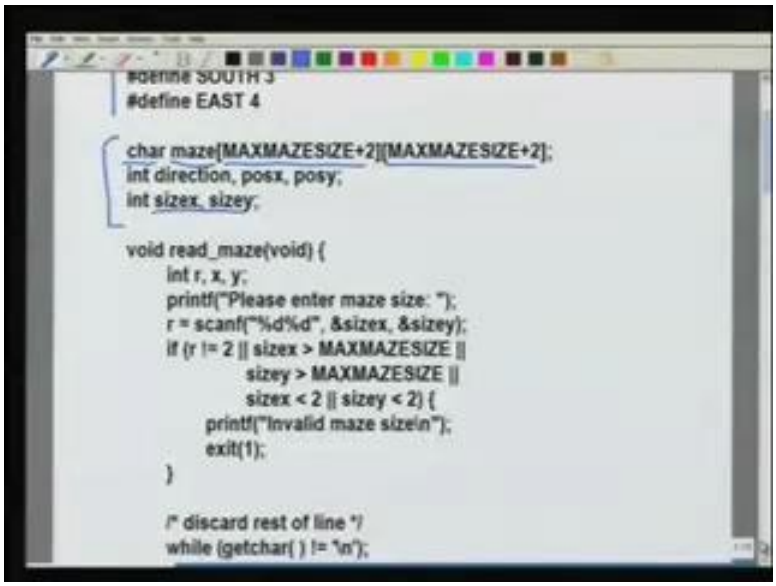
(Refer Slide Time: 12:02)



So, we have def defined this to have the value 20 meaning that the maximum maze size can be 20 by 20, and then we have defined these four constants for the four directions the north west south, and east they could have been defined to any numbers as long as they are distinct numbers. Now, these are the global variables that we had already identified for the program the maze is an is a two dimensional array of characters of size max maze plus max maze size plus 2 into max maze size plus 2.

The direction is the current direction we are facing which will be one of north west south, and east posx, and posy are the coordinates of the current position within the maze, and sizex sizey are the actual maze size in the x, and the y direction. So, let's now, look at the function read maze that is the first function. So, we first ask the user to enter the maze size, and read the sizes in the two direction sizex, and sizey.

(Refer Slide Time: 12:58)



Note that we store the return value of scanf in a local variable r, and now, we need to check whether the user supplied input for the maze sizes is correct or not. So, if the value of r is not equal to 2; that means, it did not really enter 2 integers, and therefore, an error message will be printed or if the x size or the y size is more than the max maze size, then the size here specified for the maze is too large, and we cannot handle that.

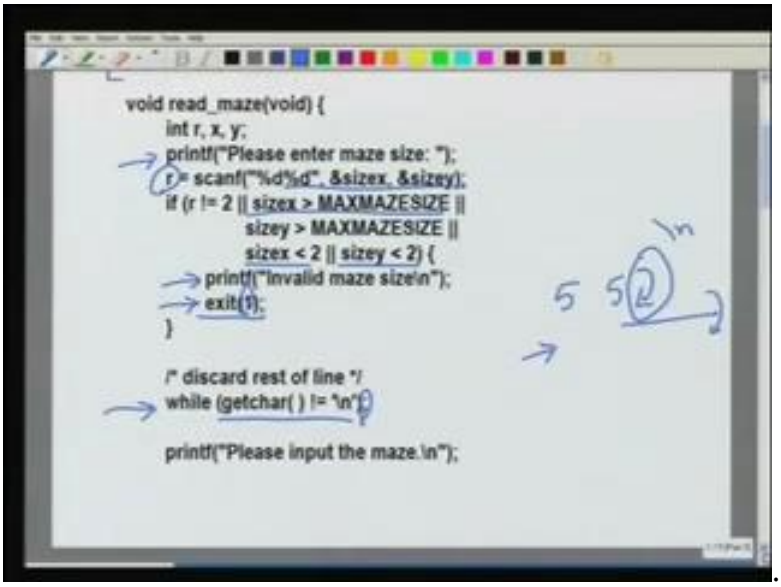
So, we again we should give an error message or if the maze size in either direction is too small less maze of size less than two does not really make sense, then also we should print an error message. So, in any of these circumstances we print the error message that the maze size that was entered was invalid, and then we use a library function exit here.

Now, this is a function that we are seeing for the first time. So, this exit function is a library function, which causes the program execution to terminate immediately. Note that this is different from the return statement in that the return statement causes the currently executing function to immediately return, but the, but calling the exit function causes the entire program to terminate immediately.

The argument to the exit function is an integer which we have specified as one here. So, this argument to exit function is a kind of exit status of the program, and usually the convention is that if the program terminates successfully, then it should exit with an exit status of 0, otherwise if it terminates to some kind of error conditions, then it should exist within with a non 0 exit status.

So, since in this case we exiting, because of an error in the input we are supplying a non 0 value for the exit status. So, after having read the size of the maze we will need to read the actual maze, but before doing that we need to be careful, because the first line of the input that the user entered contain the two integers, and at least the return character or the end of line character after that. Now, we are going to use the getchar function for reading the configuration of maze that is for each position in the maze, whether it contains a open space or part of a wall.

(Refer Slide Time: 17:08)

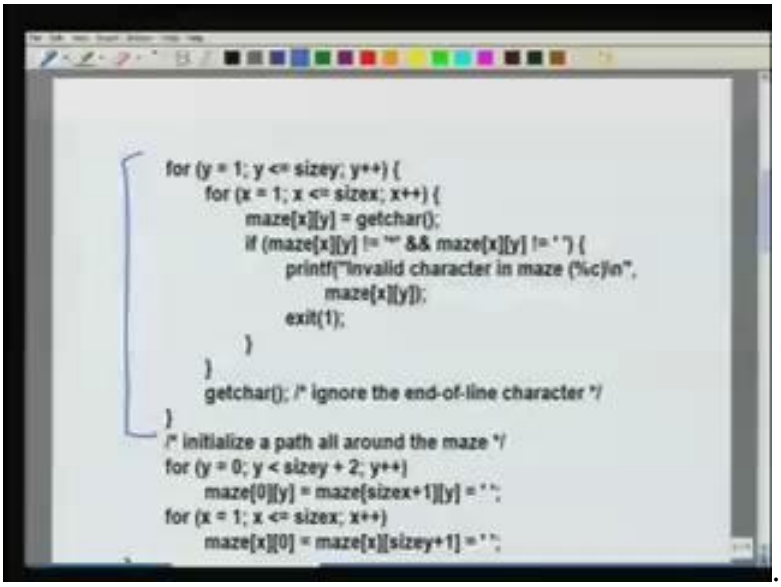


So, in the getchar library function recall that it does not skip any character in the input. So, suppose the user entered 5, and 5 for the size of the maze, and then he entered the return character. Now, if we now, read the next character we will see the new line character as the value returned by the getchar function. Now, this is clearly not part of the maze itself.

So, what we need to do is that whatever the user entered on this line after the second integer, and till an including a new line character that is needs to be skipped. So, that is precisely what this simple, while loop is doing note that the body of while loop is empty it has just this empty statement in it. So, the body is not doing anything all the action is actually happening in the condition itself which has the side effects of reading a character, because it is calling the getchar function.

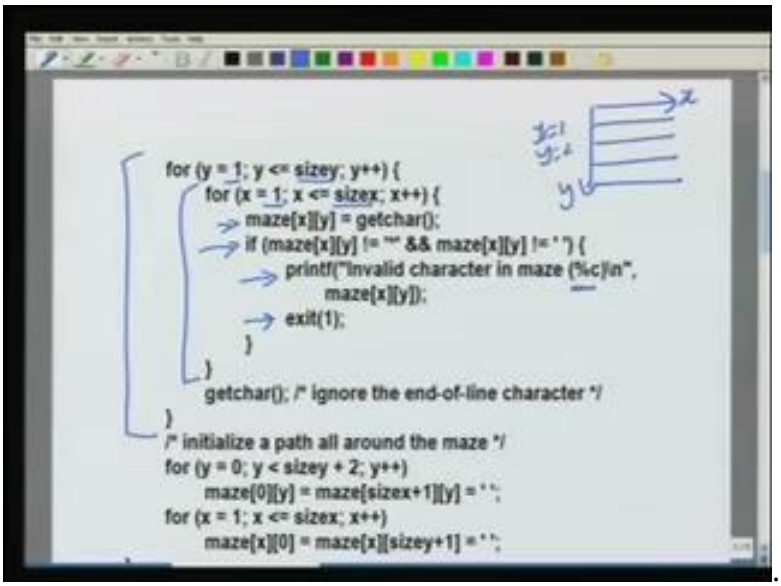
So, as long as we read characters as long as we do not see a new line character, and we stop reading characters, when we have seen a new line character, and the effect of that is that on this input first input line whatever the user entered after the second integer till the new line character all these character are read from the input, and they are discarded. So, the next character that we read from the input will be the first character entered by the user on the next line. So, after here after doing that we prompt the user to start in putting the maze, and this big loop reads the maze.

(Refer Slide Time: 17:18)



Now, how are we going to read the maze. I said we are going to use the getchar charact getchar function to read individual characters in the maze, recall that our x, and y directions were like this. So, the user is going to give the maze configuration in this fashion that is for he will first supply the entire row for y equal to 1, and then the row for y equal to 2, and so on so forth.

(Refer Slide Time: 19:00)



The y equal to 0 row in the array we will reserve for the path that we are going to create around the maze. So, for each successive value of y from one till sizey we have to read the entire row of maze characters, and the number of characters where is sizex. So, for x from one to sizex we read the maze character at the position x y using the getchar function, and now, again make sanity check from the input that we have just received. So, the maze character that we have just seen should either be a star character indicating a wall or it should be a blank space character indicating an open path.

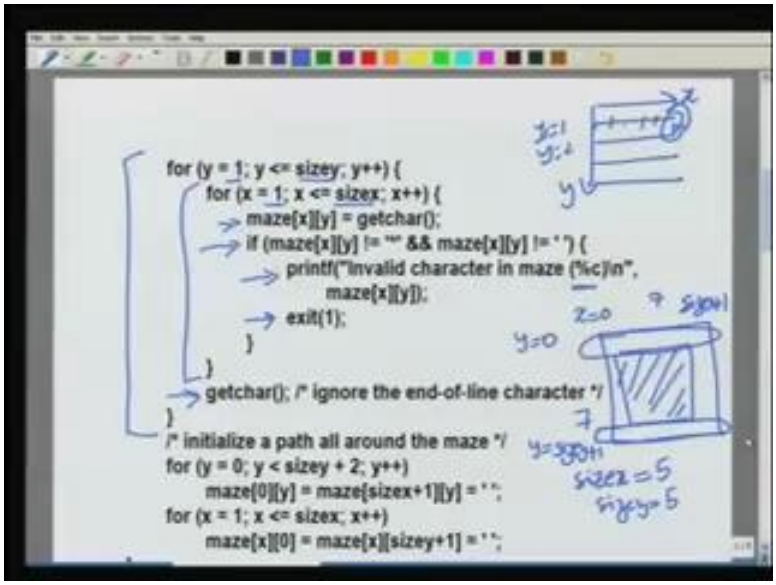
So, if the character that we just saw is neither star nor the blank character, then again we print an error message that an invalid character in the maze was entered by the user, and we specify in the bracket what character was found to be invalid, and then we exit from the program. So, once this inner loop terminates we have read all the characters in the in a particular row, and then at the end user of course, terminated this line by by an hitting an enter key. Which means a new line character is also there.

So, now, if this immediately start reading the next row, we the first character that we will read will be the new line character. Which of course, would be an invalid character.So, what we need to do is to discard again this new line character, and that we are doing by calling getchar one more time to discard the end of line character at the end of the line.

So, after having read the maze now, our array is of size lets say seven by seven if the sizex was 5, and sizey was also 5, and what we have done is that we have read this part

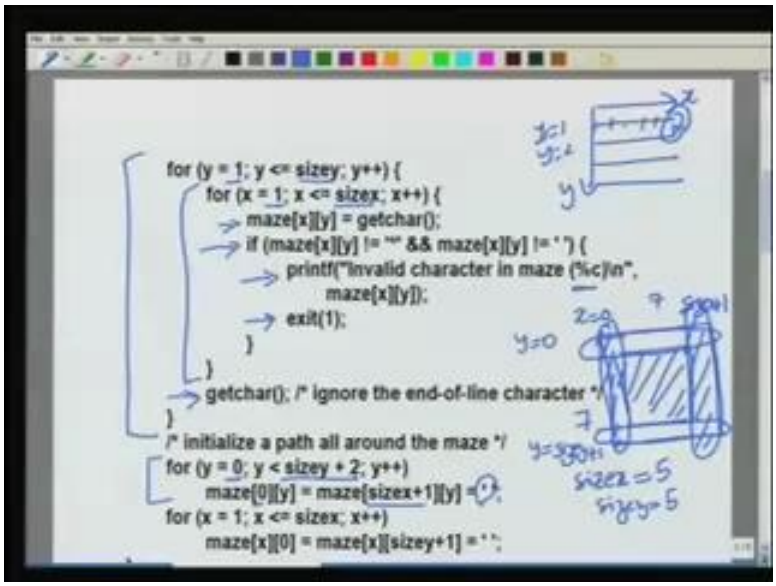
of the array from the user, and this part of the array outside this has to be initialized to a to a open path.

(Refer Slide Time: 20:42).



Now, note that this part of the open path will have the value y equal to zero, and x will range from 0 to sizex plus plus one, and similarly this portion of the path will have y equal to sizey plus one, and x again will range from 0 to sizex plus 1, and similarly these two paths will have the same y, but will have the same x, but different values of y. So, this first loop is initializing these paths in north north south direction from the two sides of the maze. So, essentially what we are doing is for x is equal to 0, and for x equal to sizex plus 1, we are initializing all elements of the maze to the blank character for y from zero to sizey plus 1.

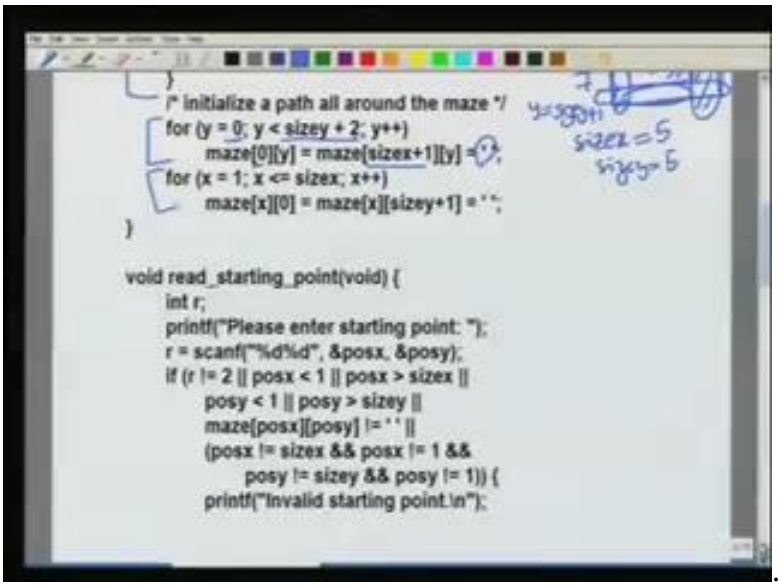
(Refer Slide Time: 21:18)



Note that the loop runs till y is less than sizey plus two which means that y is less than or equal to sizey plus 1, and then this loop essentially initializes these two parts of the path. So, now we have created the open path around the maze also there is nothing else to be done in this function. So, the function finishes note that there is no return statement.

Return statement as we had discussed is not needed for functions that do not have any return value when the function body finishes, then the function automatically returns. So, here is the next function that we identified namely read staring point. So, again it has no parameter, and has no return value it will read the starting point, and store the starting point in the variable posx posy which are declared to be global variables.

(Refer Slide Time: 22:03)

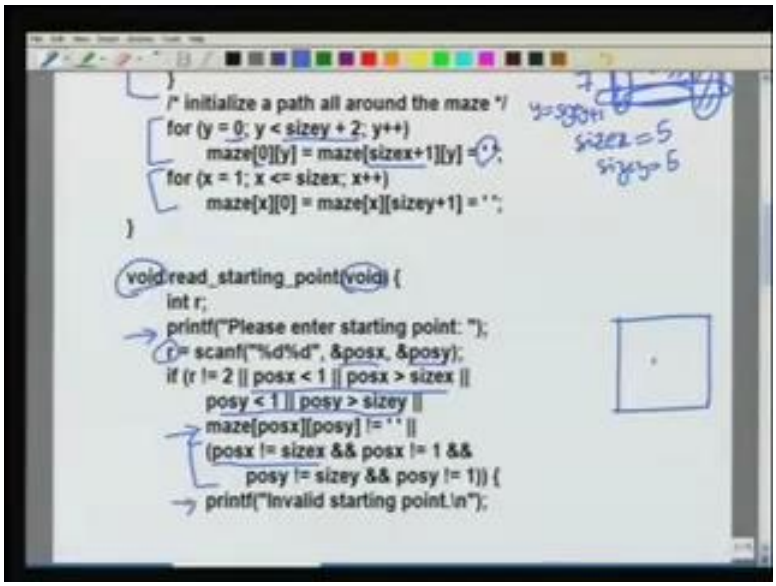


So, again it ask the user to enter the starting point, and reads two integer into posx, and posy which are global variables, and again stores the return values of scanf into a local variable r. So, if the if the value of r is not equal to two or if the starting position is not a valid starting position, then we want to print an error message, and exit from the program.

So, what do we mean by start by a valid starting position well well the starting position must be on the boundary of the maze. So, first of all what we are going to check is that posx, and posy have valid values that is posx must be at least 1, and pos posx must be no more than sizex. And similarly posy must be at least 1, and posy must be no more than sizey, and, and then we are checking that the starting point that the user has specified at that point in the maze there is an open space.

Because if at this place there is no open space, then this cannot be a starting point, and then this last check ensures that the starting point is indeed on the boundary of the maze. So, a point is on the boundary of the maze if either x the the value of x is one or sizex or the value of y is 1 or sizey. So, if the value of x is 1, then it is on this border of the maze somewhere if the value of x sizex, then the starting point is somewhere on this side of the maze.

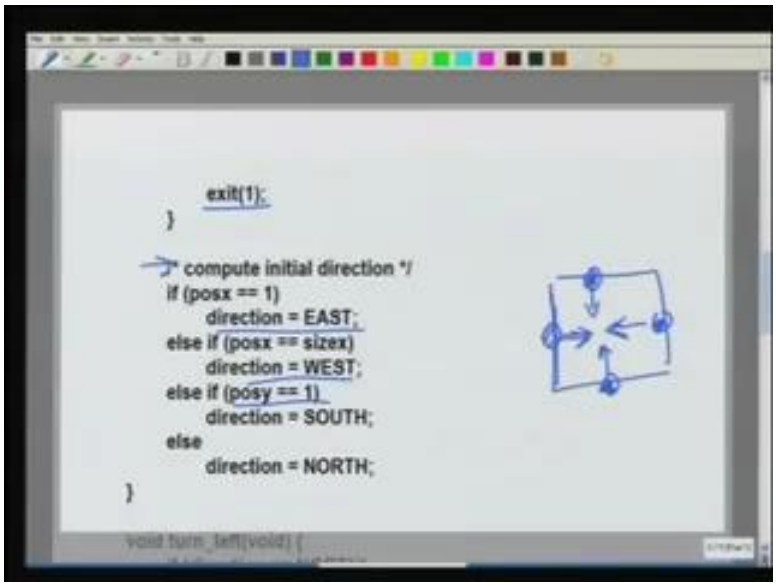
(Refer Slide Time: 24:24)



If the value of y is 1, then the position is somewhere on this line, and if the value of y is sizey, then the starting position is on on this edge, but if it is on neither of these four edges, then this starting point specified is somewhere inside the maze, and that is how it is not permitted. So, therefore, if posx is not equal not sizex, and posx is not equal to one, and posy is not equal to sizey, and posy is not equal to one, then the starting point that the user has specified is again invalid. So, in that case also we print the message that its an invalid starting point, and terminate the program by calling the exit function.

So, we print that it's an invalid starting point, and then we terminate the function by calling exit. So, now if we reach this point in program; that means, that the starting point entered by the user was a valid starting point. And now, we need to initialize the initial direction now, how do we initialize the direction to begin with well that is again simple.

(Refer Slide Time: 26:22)

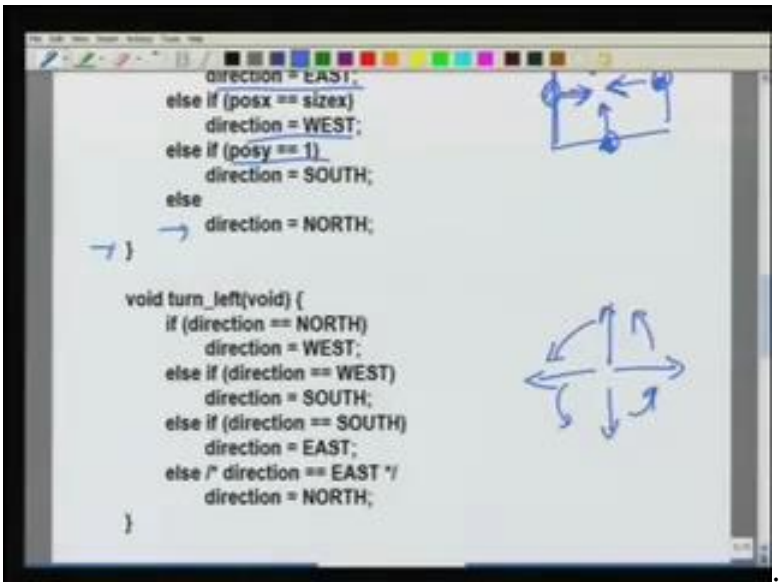


If the starting position is somewhere here, then the initial direction must be south similarly if starting position is somewhere here the initial position initial direction must be west if it is here, then it must be north, and if it is here, then it must be west. So, if posx is one; that means, the starting point is like this somewhere on the left edge of the maze, then the starting direction is east this way if posx is equal to sizex, then it is a situation like this, and the starting direction is west this way

Similarly, if posy is one, then it is a situation like this, and therefore, the direction is south otherwise it must be the case that the that the starting point is on the bottom edge of the maze, because if it is on neither of the four edges we would have not accepted. It is a starting point at all.

So, in this last case the starting point the starting direction would be north. So, after having computed the starting direction in this way the function simply returns or the initialization, we simply need to do have been done. Lets now, look at the next function which is turn left, and this is quite simple depending on the current direction just the direction changes the position with in the maze does not really change.

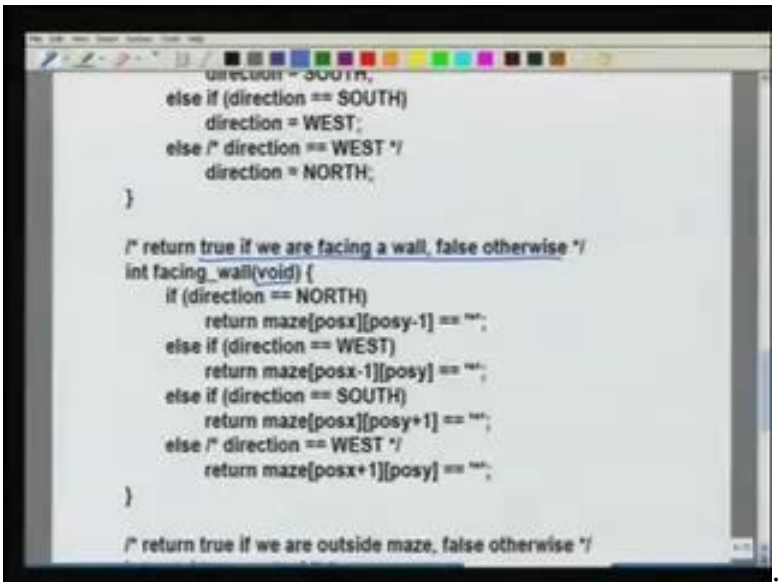
(Refer Slide Time: 27:15)



So, if right now we are facing north, then on a left turn we will start facing west. So, if the direction is not north, then the direction becomes west. Similarly, if the direction is west if we take a left turn we will start facing south, and if you were initially facing south, then on taking left turn you start facing east, and otherwise if the direction is neither north, west or south, then it must be the case that direction is east.

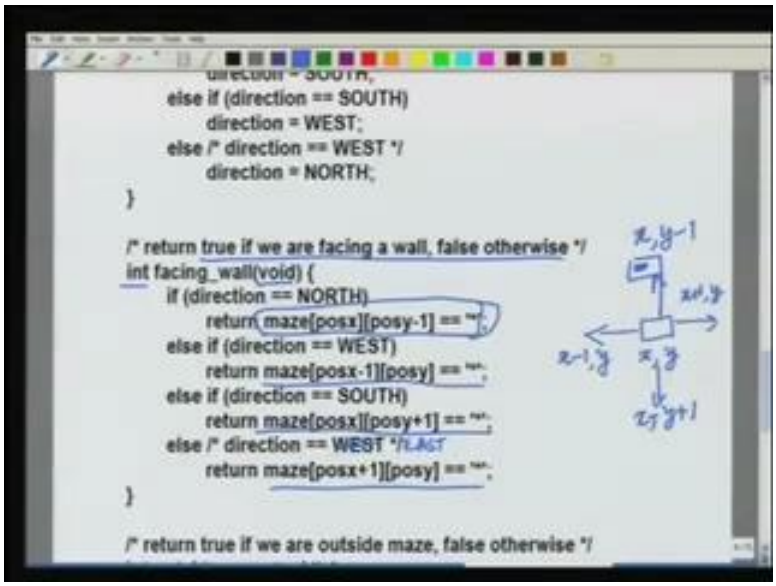
So, on a left turn the direction becomes north the turn right function is exactly similar except that from north we go to east, and so on so forth. So, the next function is to determine whether we are in the current position, and in the current direction facing a wall. So, this function is suppose to return true if we are facing a wall, and false otherwise. So, again there are no parameter, because all the information that we need is available in global variable.

(Refer Slide Time: 27:47)



But there is a return value true or false, and in C as you know there is no Boolean types. So, we use the integer type to represent Boolean. So, we have declared the return type as into now how do we determine whether we are facing a wall or not. Let us say we are facing in the north direction that is the current direction is north, then if this is the current position, then if we go one step north the current position, then the new position will become x comma y minus 1.

(Refer Slide Time: 30:20)



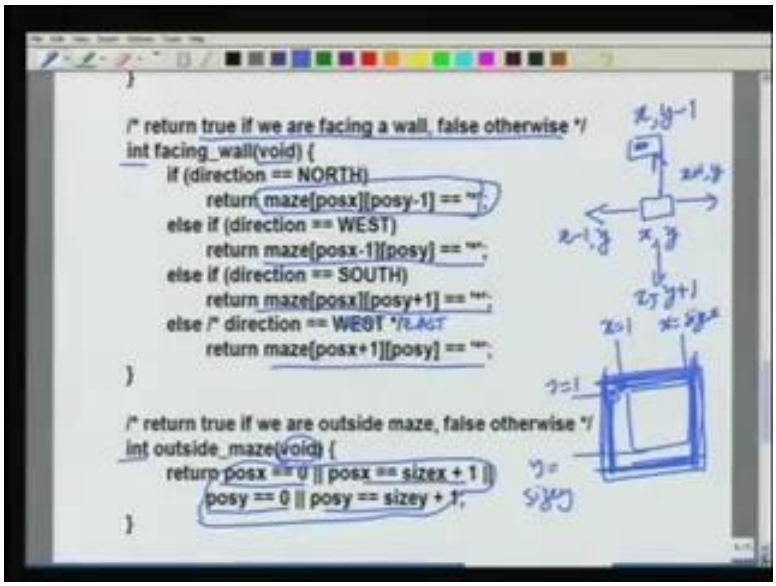
Because x will remain the same, and y will get decremented by one now, if whether we are facing a wall or not depends on whether the maze at this new position had been moved one step north from the current position cont this new position, whether it contains a wall or not. So, if maze posx posy minus one is equal to star; that means, we are facing a wall otherwise we are not facing a wall.

So, note that what we are returning is the value of this entire expression, and the value of this entire expression is true if the maze element at posx posy minus one is star, and its false, if the maze element at this position is a blank. So, that is precisely what we are suppose to return and. So, this therefore, this is the current this is the current return value similarly if the direction that we are facing is west.

Then if we move one step west from the current position, then the new position would become x minus one comma y and. So, what we return is whether maze posx minus one posy is equal to star or not. Similarly, if the direction is south, then moving one step in the south direction would lead us to x comma y plus one, and that is what we return here whether that new position contains a wall or not, and similarly as the direction is east this should be east.

So, we are facing east by moving one step we would have reached x plus one comma y and. So, in this case we return the result of checking whether at that new position the maze contains the wall or not. The next function is to determine whether the current position is outside the maze or not. So, again there are no parameter, because all the information required is available in global variable, and the return value is of type into, because they are going to return true or false.

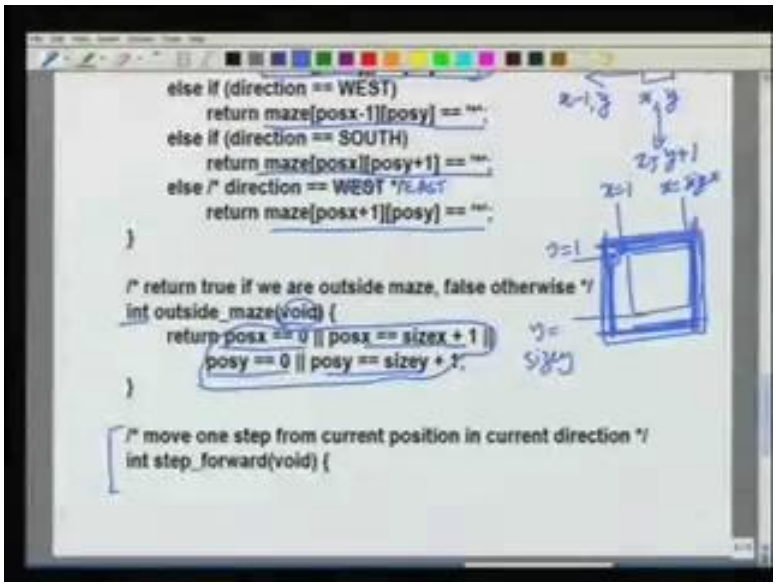
(Refer Slide Time: 32:04)



So, how do we determine whether we are outside the maze or not all we have to check is whether we are somewhere on this path that we had created around the maze or not, and so, if posx is zero note that the actual maze runs from y equal to one to y equal to sizey, and in the x direction from x equal to one to x equal to sizex. So, if the current value x is 0; that means, we are somewhere on the path or if the current position of x is sizex plus 1, then we are somewhere on this path, and if the current value of y is 0; that means, you are somewhere on this path, and if the value of y is sizey plus 1, then we are somewhere on this path.

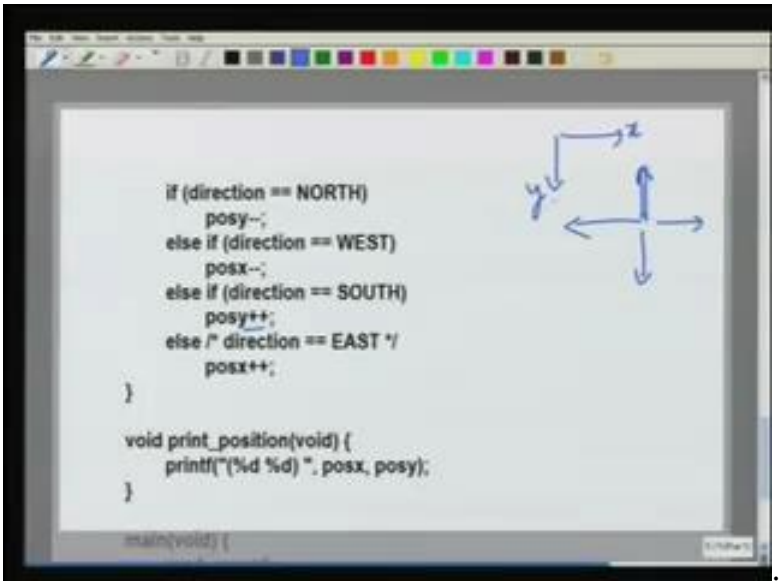
So, in all these cases we are suppose to return true, and that is what precisely this expression consisting of or of these multiple condition will do? So, if posx is equal to zero, then this entire expression whose value we are returning will evaluate to true, and similarly if posx is equal to sizex plus one, then this expression evaluates to true, and therefore, the entire expression also evaluates to true and. So, on, and in all other cases the expression representing the return value evaluates to false and. So, therefore, we end up returning the right value. So, that is almost the end of the program just need to write couple of simple functions. So, the next function is to step forward from the current position in the current direction.

(Refer Slide Time: 32:20)



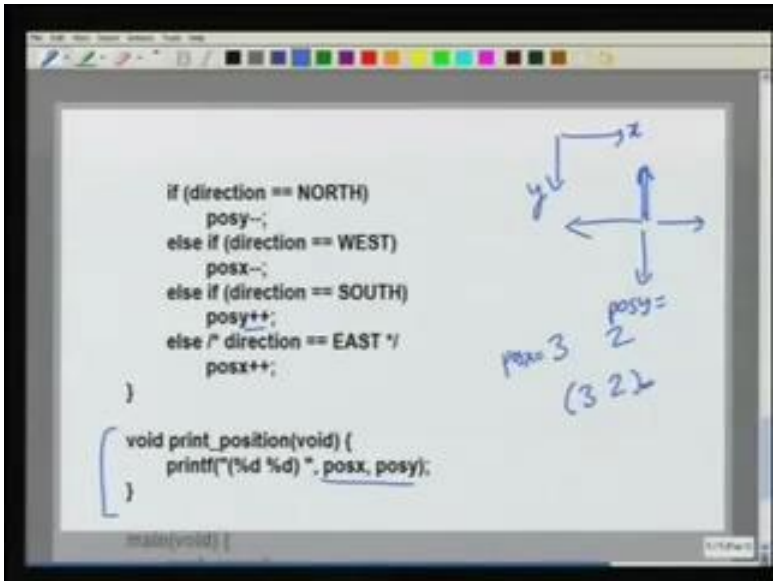
So, all we need to do is to change the value of the current position either in the x direction or in the y direction depending on whether we are facing west north south or east. So, we are facing north, then by moving one step in this direction the value of y will get decremented by 1. If we are facing west, then the value of x will get decremented by one if we are facing south, then the value of y will get incremented by one, and if we are facing east, then the value of x will get incremented by 1.

(Refer Slide Time: 33:00)



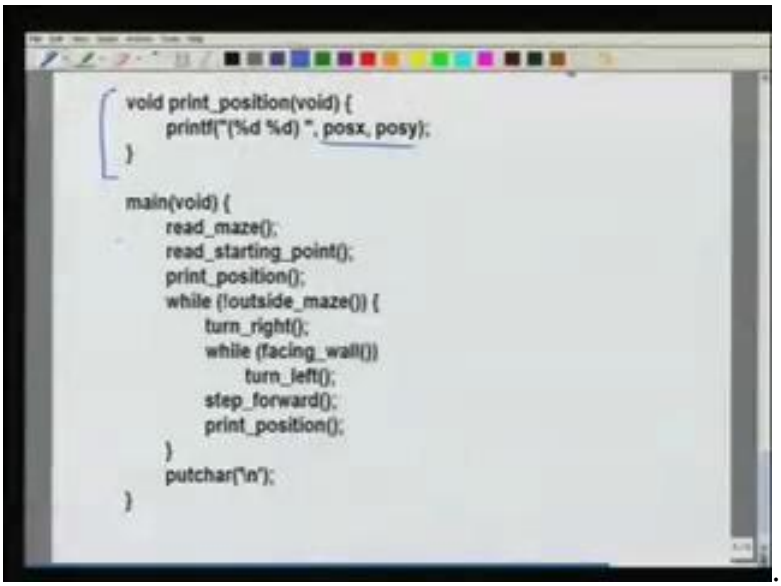
Remember that our coordinate system is like this x is increasing this way, and y is increasing southwards. So, next function print position is equally simple all it does is that it prints the current position posx posy in this notation. So, if the current position happens to be 3, and 2 that is posx equal to three, and posy equal to 2, then this function will print something like three comma two with a blank after the bracket.

(Refer Slide Time: 33:26)



So, that if we successively print multiple positions within the maze, they are separated by blank spaces. So, finally, we are in a position to write the main program which will just call all these functions that we have written.

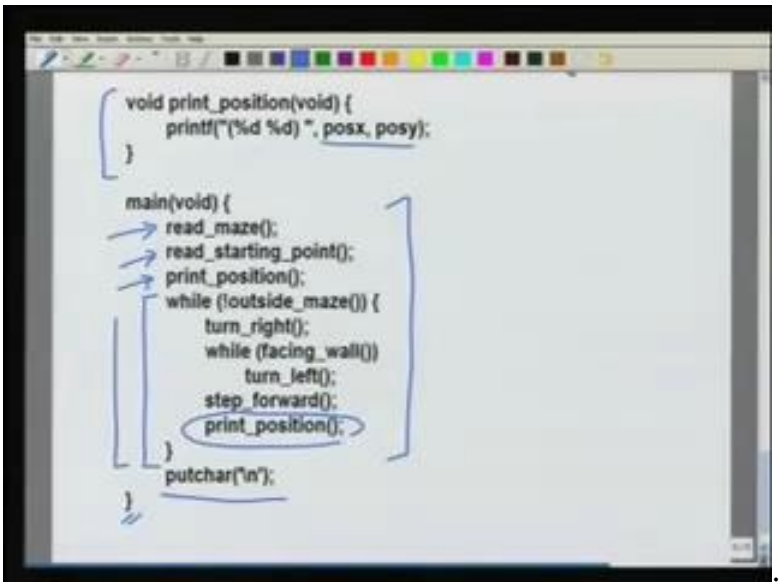
(Refer Slide Time: 33:44)



So, in the main function we first read the maze note that the main function does not really require any local variables at all. We first read the maze, then we read the starting point, then we print the current position, and then essentially this loop captures the algorithm for this problem that we have already seen. While we are not outside the maze turn right, and then while facing a wall turn left, and then step forward, and then since we have step forward we have move to a new position.

So, we print that position as well. So, at the end of this loop the entire path has been printed, and then we finally, print a new line character. So, that the cursor come to the next line, and that is all that the main program needs to do? You can see that once the functions have been written the main program is really simple, and somebody on reading this immediately understand what are algorithm for solving. This particular problem is…

(Refer Slide Time: 34:45)



So, that the end of today's lecture, in the next lecture we will start by discussing another important tool in programming, and that is the notion of pointers.