

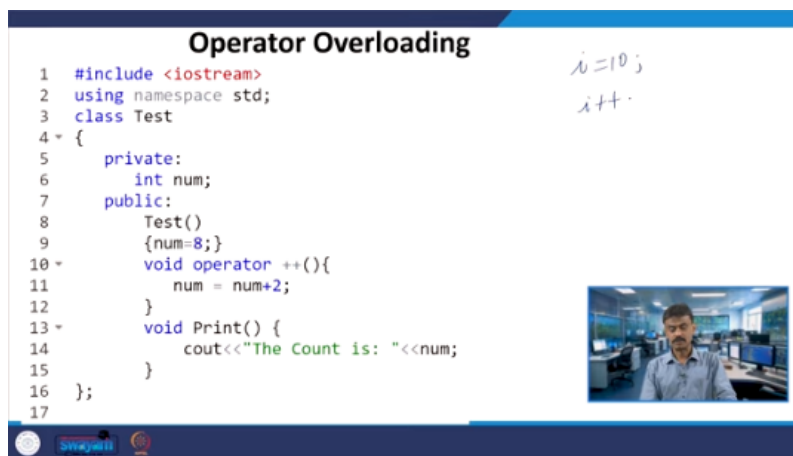
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture17

Lecture 17: Overloading - Operator and Constructor

Thank you. Welcome to lecture 17 polymorphism. So, in the last lecture we talked about function overloading right also I discuss about overriding right. So, that means we had seen compile time polymorphism and runtime polymorphism and in today's lecture I will talk about the operator overloading. So, what do you mean by operator overloading?

So, we know that right suppose I write I plus plus I is equal to 10 right. And then I write I++. Correct. So I is equal to 10. And then I write I++.



```
Operator Overloading
1  #include <iostream>
2  using namespace std;
3  class Test
4  {
5  private:
6      int num;
7  public:
8      Test()
9          {num=8;}
10     void operator ++(){
11         num = num+2;
12     }
13     void Print() {
14         cout<<"The Count is: "<<num;
15     }
16 };
17
```

So we know that suppose I put C out I. We know it will print 11. So now the same++ operator. Alright. So can we use for some other purpose. Exactly like a function overloading.

Right. Function overloading what you have done. Same function name. You have used for the different purpose. So, in a similar way, I know I++ will give you I equal to I plus 1.

So, now the same concept can be used for something else, right? For different purpose, I mean to say. So, you call that as a operator overloading. So, let us say

you have one private member data called number, right? So, then you have a constructor test.

So, which is initializing number equal to 8. which is a default constructor. And here the operator overloading line number 10 to 12 operator plus plus. So this is a syntax. You use the return type void, right?

In fact, no return type void type. And then you have operator plus plus operator keyword. And then you have plus plus. You are writing like a function, right? You are exactly writing like a function, right?

So you have operator keyword and then plus plus. That means I am going to use plus plus for different purposes. So, plus plus what I am doing, number is equal to number plus 2. It is exactly, think about the function. When I am calling this, it has to invoke this particular function and assume that you have some object, right, a. a dot number initially should be 8 and it will be incremented by 2.

So, that we are going to see, how we are going to work it out on this, right. So, then we have a print statement, right. So, you have one print statement that is printing. That is a print member function that is printing the count. Count is whatever number you are getting as the output.

So, this is inside the class, the private member data number. So, that can be accessed. So, the number is being printed over here. So, now I have an object. All right.


So, the object tt. Now, if you look line number 21, I am using plus plus tt. Right? It is like you use plus plus i or i plus plus. Right?

So that means this will call the function. So which function will be called? That void operator plus plus. So this will be called. So when it is calling what is happening?

So tt is an object and when it is an object what will be the number? tt dot number. Right? So that will be assigned to 8. It is calling the default constructor.

```
18 int main()
19 {
20     Test tt;
21     ++tt; // calling of a function "void operator ++()"
22     tt.Print();
23     return 0;
24 }
25
```

++i



Correct? So it is calling the default constructor. When you are initializing an object right so when you are instantiating an object it is calling the default constructor when it is calling a default constructor number is assigned to 8 correct and then the next line number 21 plus plus tt so that means this object is calling the function or invoking the function operator plus plus right so operator plus plus number is 8 So number is equal to number plus 2.

8 plus 2 is what? 10. So number, it is becoming 10. So now you can see the tt.print. The tt is invoking the print function.

So when it is invoking the print function, let us say what is happening. So it will print the count is the number. What is the number now? It has become 10. So the 10 should be printed.

So I will get the output 10. All right. So when I run the code, here you can see the output is nothing but 10. All right. So usually, why it is called overloading?

Usually when I use this i is equal to 10 and then assume that I use plus plus i. And then the third statement I will make c out i. Right. So we know that. Right. When I am using this, assume that I do not know operator overloading. So when you use this, so you have been taught this is nothing but i equal to i plus 1.

So one value is getting incremented. Right. So now the same plus plus can I use for object. That means I am using for different purpose. So when I am calling this particular function means I mean to say when I am calling this particular statement means it is nothing but the object tt which is invoking the void operator plus plus function.

Right. So this will be invoked. That is a mean, right? So this will be invoked. Line number 10 to 12 will be invoked.

So when you are creating the object, the number is 8. Now the number will be increased to 8 plus 2, 10. And then in the print statement, this will be displayed, right? So this is what exactly happening. And you call this concept as operator overloading.

I am using the plus plus operator to overload. So you can use any operator. right for the overloading whatever operators that you have studied so you can use for overloading. Similarly in fact we have already studied right when you have assumed that in the same program you have a default constructor parametrized constructor parametrized constructor of let us say two arguments same name obviously same name constructor right it has having assumed the three arguments correct so which one will be called. So you call that as constructor overloading, right.

So now you know operator overloading, function overloading you know. So you know operator overloading. And now we are going to study constructor overloading. Right. Assume that you have the name of the class is person.

So this is a function overloading and operator overloading are nothing but static binding. Right. So let us assume you have private int age. Private member data. And then you have public there is a member function person.

Age is assigned to 17. Age is assigned to 17. Right. So that means this is a default constructor. Person is a default constructor.

Special member function. So, under this age is assigned to 17, age is equal to 17. So, this is the default constructor or you can say constructor with no arguments. The second constructor with an argument, one argument person int a, person int a and then a will be whatever you are going to pass. In the main.

```

14      // 2. Constructor with an argument
15      Person(int a)
16      { // when parameterised Constructor is called with a value the
17        // age passed will be initialised
18        age = a;
19      }
20

```



Right. So that will be assigned to H. So that is what this constructor is doing. Parameters constructor. The parameters constructor is having only one argument. All right.

So now. So we have seen up to this line number 20. All right. So I consolidated it in this particular slide. Correct.

So you have a private member data. One constructor with no argument. And another one is constructor with an argument. Up to line number 20 we had studied. Right.

So I split over here. So this I put it everything together. So now let us go line number 21. So you have one member function int get age, right? So that will return age.

So your class is over now. So now go to the main program. When you are going to the main program, you have two objects. One object is default constructor. So when you are creating an object person 1,


Right. When you are creating an object person 1. Right. So let us see what is happening. It is calling the default constructor.

Constructor Overloading

```

1 // C++ program to demonstrate constructor overloading
2 #include <iostream>
3 using namespace std;
4
5 class Person { // create person class
6     private:
7         int age; // data member
8     public:
9         // 1. Constructor with no arguments
10        Person()
11        {
12            age = 17; // when object is created the age will be 17
13        }
14        // 2. Constructor with an argument
15        Person(int a)
16        { // when parameterised Constructor is called with a value the
17            // age passed will be initialised
18            age = a;
19        }
20    }

```



So when it is calling default constructor. Right. Age will be assigned to 17. Right. When it is calling default constructor age will be assigned to 17.

Right. And the next is person 2 of 49. You have another object. Right. Person 2.

You are passing 49. Right. You are passing the value 49. 49. So, when you are passing the value 49, so let us say what will happen?


It will call the constructor with one argument, right? So, you will write this particular constructor will be called, right? Constructor with one argument will be called because you are passing 49. So, when you are passing this, this particular A, right, 49 I am passing here. So, that will be assigned to age.

So, age is what now? 49, right? So, this is what exactly happening. Age will be 49 now. So, now you are printing two statements.

See out. Person 1 age, person 1 dot get age. So, person 1 is invoking get age. When it is invoking get age, so here you go, return age. So, for the person 1, what is the age?

17. So, the first output 17 will be displayed. Line number 31, 17 will be displayed. And person 2 is invoking get age, return age. For person 2, what is the age?

```
21     int getAge()
22     { // getter to return the age
23         return age;
24     }
25 };
26
27 int main()
28 {
29     // called the object of person class in different way
30     Person person1, person2(49);
31     cout << "Person1 Age = " << person1.getAge() << endl;
32     cout << "Person2 Age = " << person2.getAge() << endl;
33     return 0;
34 }
35
```



If you are passing 49, therefore 49 will be displayed. So, if I run the code, I will get the output person 1 age is 17 and person 2 age is 49. I hope it is clear for everyone. When we are talking about the constructor and in fact, we have studied several examples, we have seen several examples and we have run the code. So, when you are creating an object, which constructor will be called will be decided based on the number of arguments.

So, you call this concept as constructor overloading, correct? I hope it is clear for everyone. So, when I run the code, I will get person 1 age is 17 and person 2 age is 49. So, let us have one case study. So, we have seen several overloading, function overloading.

Function overloading, we had seen several examples. Operator overloading, we had seen one example, right? The plus plus. And in the previous cases, the constructor overloading. So, now let us consider one particular program.

So, write a C++ program to multiply two matrices using operator overloading all right. I have two matrices let us say M1 and M2. So, now I will have resultant is also a matrix. So, can I write like this that is what the task. So, now multiplication you know two numbers you are given it will be multiplied.

So, here in this case I am going to use this star operator for some other purpose. So, that purpose is multiplication of two matrices. So, question is can it be done? Yes, it will be done, right. So, we are going to see how we are going to do this.


So, let us have the class matrix, right. So, let us have the class matrix. You have rows and columns and then you have the two-dimensional matrix in star, star, right, pointer to pointer. One pointer is pointing to row, another pointer is pointing

to column and name of the two-dimensional array is data or you call it as name of the two-dimensional pointer is data. So, now I have the constructor, two argument constructor, int r, int c. So, whatever you are going to pass, right?

So, r will be assigned to rows and c will be assigned to columns. So, the number of rows is r. And number of columns is C. So, now I will be dynamically allocating a memory data equal to right. So, data is going to be your matrix right. In fact, the two dimensional pointers pointer to pointer.

So, now data is equal to new in star rows. So, when the star is for columns right that we are going to define. So, rows number of rows you are specifying. What about the number of columns? So number of columns, so each row I have to dynamically allocate, right?

```
15 // Dynamically allocate memory for 2D array
16 data = new int*[rows];
17 for (int i = 0; i < rows; i++) {
18     data[i] = new int[cols];
19 }
20
21
22 // Function to input matrix elements
23 void inputMatrix() {
24     cout << "Enter the elements of the matrix (" << rows << "x" << cols << ")
25                                     : \n";
26     for (int i = 0; i < rows; i++) {
27         for (int j = 0; j < cols; j++) {
28             cin >> data[i][j];
29         }
30     }
31 }
32
```



So every row number of elements or index that is running from 0 to row minus 1, correct? 0 to row minus 1 and I am assigning all the columns, all right? So 0 to row minus 1, all right? I am assigning all the elements, 0th column, what is the number of elements, right? So 0th column, one particular column, one particular row, what is the number of columns?

0th row, what is the number of columns? I am putting over here. So if it is fixed it will be like rows cross columns matrix. So as a usual way the one dimensional case we have done data here we put data of I because data of 0 first when you have right let us say this is row number 1 or row number 0. Let us start with 0 index 0.

So row number 0 what is the number of columns. So since I put COLS in all the cases number of columns will be same. So in 1, what is the number of columns?

2, what is the number of columns and so on. Up to rows minus 1, what is the number of columns?


So here you are dynamically allocating the memory. That is what you are doing from line number 15 to 21. And then you take the input from the user. Input matrix. You are taking the input from the user.

So enter the elements of the matrix. That means I am entering rows and columns. All right. So row multi it is like a multiplication of kind of into columns. All right.

You are entering rows and columns. So now you are taking the input from the user for i is ranging from 0 to rows minus 1 and j is ranging from 0 to columns minus 1. You are taking the input from the user data of i comma j. So, it goes data of 0, 0 to data of rows minus 1, columns minus 1. Correct?

So, you are taking the input from the user. Now, you are displaying the matrix. Right? Whatever you are taking the input from the user. So, this is how we can display.

```
33 // Function to display the matrix
34 * void displayMatrix() const {
35 *     for (int i = 0; i < rows; i++) {
36 *         for (int j = 0; j < cols; j++) {
37             cout << data[i][j] << " ";
38         }
39         cout << endl;
40     }
41 }
42
43 // Overloading the * operator for matrix multiplication
44 * Matrix operator*(const Matrix& other) {
45 *     if (cols != other.rows) {
46         cout << "Matrix multiplication is not possible with
47             these dimensions." << endl;
48         exit(1);
49     }
50 }
```



Right? So, this is how we can display. So, now I have the operator overloading. Operator star, the return type is matrix. Right?

So, we know that matrix is nothing but? the class, correct? So, here what we are doing? So, you have the operator overloading, correct? So, operator which you are going to overload star pausing the matrix, right?

Matrix, the address other, right? It is exactly like when you write `int address Rn` equal to `n`, right? Assume that `n` equal to 10, right? You can try this program. So, what will happen to `cout Rn`?

Any guess? So, what will happen? So, `Rn` is also equal to 10, right. So, when you are creating a variable `n`, When you are initializing a variable, let us put `int n` equal to `n`. So this is nothing but `n`. It is taking the value 10.

And what about this address `n`? Address `n` may be it is locating at some 4002, which is the extra decimal number. Now when I write like this, `int address rn` equal to `n`, address of `rn` is also same as 4002. That is the meaning. So when I do `cout rn`, star of address `rn` is nothing but `rn`.

So when I dereference it, Right. I mean to say dereference means I am going to find out what is the value at 4002. Where 4002 is the address. So I will get `Rn` is also equal to 10.

Right. So that is in a similar way we are writing here. Matrix address other. Matrix address other. So now when the columns not equal to other dot row.

So that means one matrix of the column let us say `M1`. `M1` matrix of column not equal to `M2` matrix of rows. That is what the exact meaning here. So, let us study about the object here in this particular program. Then you are printing the matrix multiplication is not possible.


Yes, the dimension has to be matched. The first matrix column should be equal to second matrix rows. If it is not equal, then matrix multiplication is not possible. This is what exactly you are telling. right?

Matrix multiplication is not possible otherwise, right? So, let us see what exactly we are going to do, right? So, here create a new matrix to store the result, right? So, that means result you are passing rows other dot columns, right? Other matrix columns.

```

51 // Create a new matrix to store the result
52 Matrix result(rows, other.cols);
53
54 // Multiply matrices
55 for (int i = 0; i < rows; i++) {
56     for (int j = 0; j < other.cols; j++) {
57         result.data[i][j] = 0;
58         for (int k = 0; k < cols; k++) {
59             result.data[i][j] += data[i][k] * other.data[k][j];
60         }
61     }
62 }
63
64 return result;
65 }

```



Result is rows and other matrix columns. One matrix rows and another matrix column. So, that means when you are creating this particular object by passing two arguments right. So, here you go you are passing R and C. So, there it is called you have the result of rows comma other columns.

So, R will be rows and C will be other dot columns correct. So, that is what exactly happening. So, this is the initialization right. So, now I am going to multiply two matrices right. So, here I is running from 0 to rows minus 1


And J is running from 0 to other columns minus 1. Right? So, initially I initialized all the data to 0. Right? So, you have rows into columns.

R cross C. R stands for row and C stands for column. So, R cross C. Correct? So, initially this R cross matrix I put everything 0. And then matrix multiplication what you are doing? The row.

```

51 // Create a new matrix to store the result
52 Matrix result(rows, other.cols);
53
54 // Multiply matrices
55 for (int i = 0; i < rows; i++) {
56     for (int j = 0; j < other.cols; j++) {
57         result.data[i][j] = 0;
58         for (int k = 0; k < cols; k++) {
59             result.data[i][j] += data[i][k] * other.data[k][j];
60         }
61     }
62 }
63
64 return result;
65 }

```



$$\begin{bmatrix} 0 & 0 & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}_{R \times C}$$

You are taking the row. For example, first element. Multiply with all the column elements. And then add it. That is what exactly you are doing.

So when k is running from 0 to columns. Minus 1. So the data of I k. One matrix. That is the meaning over here. Multiply with other dot data k of j. So you can do this.

So in one of the matrices. You are accessing other dot data. Another one data of I comma k. So, sooner or later we are going to talk about this pointer, correct. So, when I want to print this particular case, what I will do?

Because left hand side you have result dot and then the right most other dot, whereas here you do not have. So, suppose I use this right this let us say arrow operator or dot operator then this will work. So, sooner or later we are going to talk about that this concept of this pointer correct. So, you are component wise you are doing the multiplication component wise in the sense first row let us say first element I am taking all the columns will be multiplied and then you are adding. So, the resultant will be result of 0 0 and then go for result of 0 1 result of 0 2 and so on.

Okay. So, then what will happen? All the columns rows into columns will be filled. Then you are returning the result. Right.

So, this will be filled and you are returning the result. So, now I have released the memory. I have to release the memory. So, I call the destructor. For i is equal to 0, i less than rows, i++, deallocate data of i, right.

```
66
67 // Destructor to free dynamically allocated memory
68 ~Matrix() {
69     for (int i = 0; i < rows; i++) {
70         delete[] data[i];
71     }
72     delete[] data;
73 }
74 };
75
76 int main() {
77     // Define the dimensions for the first matrix
78     int r1, c1, r2, c2;
79     cout << "Enter the number of rows and columns for the first matrix: ";
80     cin >> r1 >> c1;
81
82     // Define the dimensions for the second matrix
83     cout << "Enter the number of rows and columns for the second matrix: ";
84     cin >> r2 >> c2;
85
```



So, initially I am doing 0 to rows and then I am completely deallocating the data. So, the data is the name of the matrix that we have used, right, data of i, k. So, here what we are doing, we are deallocating the memory. So, now you have R1, C1 and R2, C2, row 1, column 1 matrix, row 2, column 2 matrix. Take the input from the user R1 and C1, right? Take the input from the user row 1 and column 1.

Similarly, take the input from the user row 2 and column 2, right? So, assume that we are taking these two input, right? Assume that we are taking these two as the input. So, when you are having two objects, right? When you are having two objects.


So, let us say R1 and C1, correct? So, when you are having two objects, right? Matrix 1 and matrix 2. You have R1 and C1. Then R2 and C2.

Alright. So that means let us assume I am taking the input from the user. Alright. So let us say 10 and 5. Alright.

And here 5 and 10. That means the resultant I have to get 10 cross 10 matrix. So one is 10 cross 5 matrix. Another one is 5 cross 10 matrix. Correct.

So this what will happen when I am passing this R1 and C1. Matrix 1 the object I am So, when I am creating this object, your parametric constructor will be called. And now your rows is R1 and columns is C1. Correct?

```
86 // Create two matrix objects
87 Matrix matrix1(r1, c1);
88 Matrix matrix2(r2, c2);
89
90 // Input matrix data
91 matrix1.inputMatrix();
92 matrix2.inputMatrix();
93
94 // Multiply matrices using overloaded * operator
95 Matrix result = matrix1 * matrix2;
96
97 // Display the result matrix
98 cout << "Resultant matrix after multiplication:\n";
99 result.displayMatrix();
100
101 return 0;
102 }
```





So, that is for the first matrix. So, memory will be allocated. Everything will be done. Correct? So, input matrix also we can display.

Right? So, now I am creating the second object. R2 C2 will be my second object. Correct? I mean R2 with R2 C2 is my second object.

That means row 2, column 2. Right? So, the same. It will be calling the parametrized constructor matrix 2 and then the values will be assigned. So, the rows and columns for second matrix is R2 and C2, alright.

Next one what I am doing, I am taking the input matrix, alright. That means the object 1, the matrix 1 is invoking input matrix, right. It is invoking input matrix. So, when it is invoking input matrix, so you can see over here, alright. So, you are taking the input from the user.

```
15 // Dynamically allocate memory for 2D array
16 data = new int*[rows];
17 for (int i = 0; i < rows; i++) {
18     data[i] = new int[cols];
19 }
20
21
22 // Function to input matrix elements
23 void inputMatrix() {
24     cout << "Enter the elements of the matrix (" << rows << "x" << cols << ")
25     : \n";
26     for (int i = 0; i < rows; i++) {
27         for (int j = 0; j < cols; j++) {
28             cin >> data[i][j];
29         }
30     }
31 }
32
```



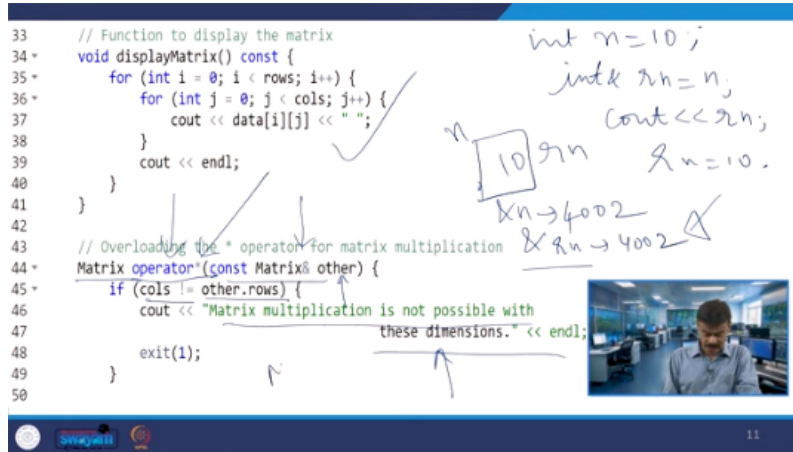
So, let us say 2 by 2. So, you can give 4 values. It will take. If you look at the code, this will take. Then the important point is line number 95.

So here only you are calling this is overloading. Matrix 1 will be multiplied by matrix 2, right? So that means you may have the operator, right? So that will go over there, correct? So it will come over here, operator star, right?

You are passing two matrices, right? It is exactly, it will work like this. If you recall, we have done addition of two complex numbers or multiplication of two complex numbers, right? One you keep as it is, another one is, that means one will invoke, another one you are passing. The same thing, similar thing you are doing over here.

It is equivalent to similar thing. M1 star M2, correct? So, M1 You are looking columns and that will be comparing with other. So that means M1 is invoking the operator star.

That is the meaning M1 star M2. M1 is invoking the operator star and M2 you are passing. When you are passing M2 that is called other. Right. So it is checking the rows and columns.



```

33 // Function to display the matrix
34 void displayMatrix() const {
35     for (int i = 0; i < rows; i++) {
36         for (int j = 0; j < cols; j++) {
37             cout << data[i][j] << " ";
38         }
39         cout << endl;
40     }
41 }
42
43 // Overloading the * operator for matrix multiplication
44 Matrix operator*(const Matrix& other) {
45     if (cols != other.rows) {
46         cout << "Matrix multiplication is not possible with
47             these dimensions." << endl;
48         exit(1);
49     }
50 }

```

Handwritten notes on the right side of the code editor:

- `int n=10;`
- `int& n=n;`
- `cout << n;`
- `n=10.`
- A box around the number 10 with `n` written next to it.
- `n → 4002`
- `&n → 4002`

Arrows point from the handwritten notes to the corresponding code lines. A small video inset in the bottom right corner shows a person speaking.

Columns of M1 and rows of M2 will be checked. If they are not equal. Then you cannot do matrix multiplication. That is what you are doing this. That is what you are informing this.

In this see out statement. And here right you have created one result matrix. And then here you are doing the multiplication. So initially keep all the data of IJ 0 0 0. And then here you are doing the matrix multiplication.

K equal to 0. K less than columns. K plus plus. As I said every row. For example A 0 0 if I take.

The other matrix column will be multiplied. Exactly it is doing this. You have the index ik and here you have kj, right? And then initially you have initialized to 0. So, whenever you are doing this component wise multiplication and then you are adding the resultant will be there, right?

So, then you are returning the result. You understand? So, one is you are passing The rows means simply rows when you are writing means m1 because we have m1 star m2. So rows of m1.


```

51 // Create a new matrix to store the result
52 Matrix result(rows, other.cols);
53
54 // Multiply matrices
55 for (int i = 0; i < rows; i++) {
56     for (int j = 0; j < other.cols; j++) {
57         result.data[i][j] = 0;
58         for (int k = 0; k < cols; k++) {
59             result.data[i][j] += data[i][k] * other.data[k][j];
60         }
61     }
62 }
63 return result;
64 }
65

```

Handwritten annotations on the code:

- A handwritten matrix $\begin{bmatrix} 0 & 1 & 2 \\ \dots & \dots & \dots \end{bmatrix}$ with dimensions $r \times c$ is shown next to line 52.
- Arrows point from the variables `rows`, `other.cols`, and `cols` in the code to their respective values in the matrix dimensions.
- The `return result;` statement is circled.



Rows of m1 is invoking star and then I am passing m2. So that means when I am passing m2 means other dot column is nothing but m2 dot columns. That is the meaning. Alright. m2 dot columns.

And then the rest of the things are very clear and then you are returning the result. So that means when I am calling this function. When I am doing this operation let us say operator overloading matrix 1 matrix 2 will be multiplied and the result will be stored over here. So, then you will have the resultant matrix after multiplication right. So, result dot display matrix right.

So, display matrix function will be called. So, here you go you have the resultant matrix M1 and M2 multiplied the result matrix you are displaying right. what we will do in the next class we will give the input one by one and then we will run this particular code. So, I will start with the matrix multiplication how we are going to give the input and then we will get the output. So, with this I am concluding this lecture.

Thank you very much.