

```

1  #include <iostream>
2  #include "row.h"
3  using namespace std;
4
5  // parameterized constructor
6  Row::Row(int length)
7  {
8      if (length <= 0)
9      {
10         throw std::out_of_range("The length has to be greater than 0");
11     }
12     this->length = length; // this->length is making the length for the Row, while length
is the length that is input
13     row_data = new double[length];
14     clear();
15 }
16
17 // copy constructor
18 Row::Row(const Row &from)
19 {
20     length = from.length;
21     row_data = new double[length];
22     for (int i = 0; i < length; i++)
23     {
24         row_data[i] = from.row_data[i];
25     }
26 }
27
28 // destructor
29 Row::~~Row()
30 {
31     delete[] row_data;
32 }
33
34 // access operator (const)
35 double Row::operator[](int column) const
36 {
37     if (column < 0 || column >= length)
38     {
39         throw out_of_range("Column must be >= 0 and < length");
40     }
41     return row_data[column];
42 }
43
44 // access operator (non-const)
45 double &Row::operator[](int column)
46 {
47     if (column < 0 || column >= length)
48     {
49         throw out_of_range("Column must be >= 0 and < length");
50     }
51     return row_data[column];
52 }
53
54 // assignment operator
55 Row &Row::operator=(const Row &rhs)
56 {
57     if (this != &rhs)
58     {
59         length = rhs.length;
60         delete[] row_data;
61         row_data = new double[length];
62         for (int i = 0; i < length; i++)
63         {
64             this->row_data[i] = rhs.row_data[i];
65         }
66     }
67     return *this;
68 }
69
70 // clear row data
71 void Row::clear()
72 {
73     for (int i = 0; i < length; i++)
74     {
75         row_data[i] = 0;
76     }
77 }

```

```

1  #ifndef row_h
2  #define row_h
3  class Row{
4      public:
5          /* Parameterized constructor
6           * Takes in length and creates a row matrix with values cleared
7           * to zero
8           * Should verify length > 0
9           */
10         Row(int length);
11
12         /* Copy constructor
13          * Create a new row matrix with the same size and values as the
14          * from matrix
15          */
16         Row(const Row& from);
17
18         /* Destructor
19          * Correctly delete any heap memory
20          */
21         ~Row();
22
23         /* Access operator (const version)
24          * Allow access to row matrix data
25          * Should return an exception if column is too large
26          */
27         double operator[](int column) const;
28
29         /* Access operator (non const version)
30          * Allow access to row matrix data
31          * Should return an exception if column is too large
32          */
33         double& operator[] (int column);
34
35         /* Assignment operator
36          * 1. Check if two sides are the same object
37          * 2. Delete the current row matrix
38          * 3. Create a new row matrix with the same size and values as
39          *    the rhs matrix
40          */
41         Row& operator= (const Row& rhs);
42
43         /* Clear all data values to zero
44          */
45         void clear();
46     private:
47         // Row matrix data
48         double * row_data;
49         // Size of row matrix
50         unsigned int length;
51 };
52 #endif

```

```

1  #include <iostream>
2  #include "row.h"
3  using namespace std;
4  int main()
5  {
6      // test constructor
7      Row R1(3);
8      cout << "R1:" << R1[0] << " " << R1[1] << " " << R1[2] << endl;
9      // test setting row data
10     R1[0] = 0;
11     R1[1] = 1;
12     R1[2] = 2;
13     Row R3(4);
14     R3[0] = 3;
15     R3[1] = 4;
16     R3[2] = 5;
17     R3[3] = 6;
18     cout << "R1:" << R1[0] << " " << R1[1] << " " << R1[2] << endl;
19     // test copy constructor
20     Row R2(R1);
21     cout << "R2:" << R2[0] << " " << R2[1] << " " << R2[2] << endl;
22     // test assignment operator when left side is larger
23
24     cout << "R3:" << R3[0] << " " << R3[1] << " " << R3[2] << endl;
25     R3 = R2;
26     cout << "R3:" << R3[0] << " " << R3[1] << " " << R3[2] << endl;
27     // test self assignment
28     R3 = R3;
29     cout << "R3:" << R3[0] << " " << R3[1] << " " << R3[2] << endl;
30     // test assignment operator when right side is larger
31     Row R4(4);
32     R4[0] = 7;
33     R4[1] = 8;
34     R4[2] = 9;
35     R4[3] = 10;
36     cout << "R4:" << R4[0] << " " << R4[1] << " " << R4[2] << " " << R4[3] << endl;
37     R3 = R4;
38     cout << "R3:" << R3[0] << " " << R3[1] << " " << R3[2] << " " << R3[3] << endl;
39     cout << "R4:" << R4[0] << " " << R4[1] << " " << R4[2] << " " << R4[3] << endl;
40     // test const access operator
41     const Row R5 = R1;
42     cout << "R5:" << R5[0] << " " << R5[1] << " " << R5[2] << endl;
43     // test clear function
44     R1.clear();
45     cout << "R1:" << R1[0] << " " << R1[1] << " " << R1[2] << endl;
46     // test constructor is throwing exceptions correctly
47     try
48     {
49         Row R6(0);
50     }
51     catch (...)
52     {
53         cout << "Row of length 0 failed" << endl;
54     }
55     try
56     {
57         Row R7(-4);
58     }
59     catch (...)
60     {
61         cout << "Row of length -4 failed" << endl;
62     }
63     // test access operator is throwing exceptions correctly
64     try
65     {
66         R1[-1];
67     }
68     catch (...)
69     {
70         cout << "column = -1 failed" << endl;
71     }
72     try
73     {
74         R1[3];
75     }
76     catch (...)
77     {
78         cout << "column = length failed" << endl;

```

```
79     }  
80     return 0;  
81 }
```

1	<b>Table of Contents</b>						
2	1 row.cpp.....	sheets	1 to	1 ( 1) pages	1-	1	78 lines
3	2 row.h.....	sheets	2 to	2 ( 1) pages	2-	2	53 lines
4	3 main.cpp.....	sheets	3 to	4 ( 2) pages	3-	4	82 lines