Ministry of Education of Republic of Moldova

Technical University of Moldova

Deapartment of Applied Informatics

Report

Laboratory Work Nr2

at *Object Oriented Programming*

Performed by:
st.gr.FAF-161 Chirosca Ariadna

Verified by:
dr., conf.univ. Kulev Mihail

Chisinau 2017-2018

Laboratory Work nr2

Topic: Constructor- function that intialize an instance of the class

Objectives:

- To study the principles of defining and use of constructors
- To study the principles of defining and use of destructors
- To study the types of constructors

Condition of the problem:
a) To create a class named Document, that will contain information about name, subject, author of the document using dynamic memory; number of pages, date and time of the new change.To define all constructors. Conversion constructor will have as parameter the name of the document.To define functions of modyfing subject , time of the last change ...
b) To create a class named Matrix.Class contains pointer to int, number of rows and columns and a variable- code of the error .Define the default constructor , Constructor with one parameter (square matrix) and constructor with 2 parameters(rectangular matrix). Define methods:setting and getting the values(i, j). Define functions of adding, substracting, multiplication with another matrix, multiplication with a constant.Test how class works.In case of memory lack ,dimension errors, memory overflow to get the error message (PS: for this I used assert())

Short theory:

Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions also known as methods.An instantiation of a class is named *object*

Classes are defined using keyword *class*.An example of a class is ilustrated below:

```
1 class Rectangle {
2     int width, height;
3   public:
4     void set_values (int,int);
5     int area (void);
6 } rect;
```

What is constructor?
A constructor is a member function of a class which initializes objects of a class. In C++,Constructor is automatically called when an instance of the class(an object) is created.

Constructor's characteristics:

- Constructor has the same name as the class;
- Constructor don't have return type;
- Is automatically called when an object is created;
- If we do not specify a constructor, C++ compiler generates a default constructor

Types of constructors:

- Default constructor(without parameters)
- Parameterized constructor
- Copy constructor(initializes an object using another object of the same class)
- Conversion constructor

What is destructor?
Destructor is a member function which destructs or deletes an object.A destructor function is called automatically when the object goes out of scope:

- The program ends
- The function ends
- Delete operator is called etc.

Destructor's characteristics:

- Have same name as the class preceded by a tilde
- Don't take any argument and don't return anything
- There can be only one destructor in a class
- Compiler creates a default destructor

Data analysis for first problem

1)Parameterized constructor:
Document::Document(char * newTitle, char * newSubject, char * newAuthor, char * newDay, int newPages, int newLastChange)
It initializes each member of the class with a value and dynamically alocate memory for each member.
2)Copy constructor:
Document::Document(const Document & doc)
It initializes the object using another object of this class.
3)Conversion constructor:
Document::Document(char * title)
Takes as parameter only the title of the document and other members are initialized by us.
4)Default constructor:
Document()
5)"Set" functions:
$voidDocument :: setTitle(char * \_title)$
$voidDocument :: setSubject(char * \_subject)$
$voidDocument :: setAuthor(constchar * \_author)$
$voidDocument :: setDay(char * \_day)$
$voidDocument :: setNumberOfPages(int\_pages)$
$voidDocument :: setLastChange(char * \_lastChange)$
Those functions are used to set the member's value according to user preference, and beacuse we have to change all the data saved into the members of our class, we take as parameter the corresponding member.

6)Data modifier:
void Document::modifyData()
According to our answer it calls the neccesarry function and changes the data

document.h

```
#ifndef _DOCUMENT_H
#define _DOCUMENT_H

#include <iostream>
#include <cstring>

using namespace std;

class Document
{
private:
  char * title;
  char * subject;
  char * author;
  char * day;
  char * lastChange;

  int pages;

  void modifyTitle();
  void modifySubject();
  void modifyAuthor();
  void modifyDay();
  void modifyPages();
  void modifylastChange();

public:
  Document() : pages(0), lastChange(new char[strlen("noChange")
   + 1]), title(new char[strlen("noTitle") + 1]), subject(new
   char[strlen("noSubject") + 1]), author(new char[strlen("
   noAuthor") + 1]), day(new char[strlen("noDay") + 1])
  { }
  Document(char *);
  Document(const Document &);
  Document(char*, char*, char*, char*, int, char*);
  ~Document();

  void modifyData();
```

```
36    void setTitle(char*);
37    void setSubject(char*);
38    void setAuthor(const char*);
39    void setDay(char*);
40    void setLastChange(char*);
41    void setNumberOfPages(int);
42    void displayDocumentInformation();
43
44 };
45
46 #endif
```

document.cpp

```
1  #include <iostream>
2  #include <iomanip>
3  #include <string>
4  #include <cstring>
5
6  #include "document.h"
7
8  using namespace std;
9
10 Document::Document(char * newTitle, char * newSubject, char *
       newAuthor, char * newDay, int newPages, char * newLastChange)
11 {
12    title = new char[strlen(newTitle) + 1];
13    strcpy(title, newTitle);
14
15    subject = new char[strlen(newSubject) + 1];
16    strcpy(subject, newSubject);
17
18    author = new char[strlen(newAuthor) + 1];
19    strcpy(author, newAuthor);
20
21    day = new char[strlen(newDay) + 1];
22    strcpy(day,  newDay);
23
24    pages = newPages;
25    lastChange = new char[strlen(newLastChange) + 1];
26    strcpy(lastChange, newLastChange);
27 }
28
29
30 Document::Document(const Document &doc) : title(doc.title),
```

```
    subject(doc.subject), author(doc.author), pages(doc.pages),
    day(doc.day), lastChange(doc.lastChange)
{
  title = new char [strlen(doc.title) + 1];
  strcpy(title, doc.title);

  subject = new char[strlen(doc.subject) + 1];
  strcpy(subject, doc.subject);

  author = new char [strlen(doc.author) + 1];
  strcpy(author, doc.author);

  day = new char[strlen(doc.day) + 1];
  strcpy(day, doc.day);

  pages = doc.pages;
  lastChange = new char[strlen(doc.lastChange) + 1];
  strcpy(lastChange, doc.lastChange);
}

Document::Document(char * title)
{
  this->title = new char[strlen(title) + 1];
  strcpy(this->title, title);
  subject = new char [strlen("noSubject") + 1];
  author = new char [strlen("noAuthor") + 1];
  day = new char [strlen("noDay") + 1];
  lastChange = new char [strlen("noChanges") + 1];
  pages = 0;

}

void Document::setTitle(char * _title)
{
  if (title)
    delete[] title;
  title = new char[strlen(_title) + 1];
  strcpy(title, _title);
}

void Document::setSubject(char * _subject)
{
  if (subject)
    delete[] subject;
  subject = new char[strlen(_subject) + 1];
```

```
74    strcpy(subject, _subject);
75  }
76
77  void Document::setAuthor(const char * _author)
78  {
79    if (author)
80      delete[] author;
81    author = new char[strlen(_author) + 1];
82    strcpy(author, _author);
83  }
84
85  void Document::setDay(char * _day)
86  {
87    if (day)
88      delete[] day;
89    day = new char[strlen(_day) + 1];
90    strcpy(day, _day);
91  }
92
93  void Document::setNumberOfPages(int _pages)
94  {
95    pages = _pages;
96  }
97
98  void Document::setLastChange(char * _lastChange)
99  {
100   if(lastChange)
101     delete[] lastChange;
102   lastChange = new char[strlen(_lastChange) + 1];
103   strcpy(lastChange, _lastChange);
104 }
105
106 void Document::displayDocumentInformation()
107 {
108   cout << "\n";
109   cout << setw(5) << setw(15) << left << "Title: "<< setw(15) <<
        left << title << endl;
110   cout << setw(5) << setw(15) << left << "Subject: " << setw(15)
        << left << subject << endl;
111   cout << setw(5) << setw(15) << left << "Author: " << setw(15)
       << left << author << endl;
112   cout << setw(5) << setw(15) << left << "Pages: " << setw(15)
       << left << pages << endl;
113   cout << setw(5) << setw(15) << left << "Day: " << setw(15)  <<
        left <<day << endl;
```

```
114   cout << setw(5) << setw(15) << left << "Last change: " << setw
        (15) << left << lastChange << endl;
115 }
116
117 void Document::modifyTitle()
118 {
119   char * _title = new char[100];
120   cout << "Enter the new title: " << endl;
121   cin >> _title;
122
123   delete[] title;
124   title = new char[strlen(_title) + 1];
125   strcpy(title, _title);
126
127 }
128
129 void Document::modifySubject()
130 {
131   char * _subject = new char[100];
132   cout << "Enter the new subject: " << endl;
133   cin >> _subject;
134
135   delete[] subject;
136   subject = new char[strlen(_subject) + 1];
137   strcpy(subject, _subject);
138 }
139
140 void Document::modifyAuthor()
141 {
142   char * _author = new char[100];
143   cout << "Enter the new name of the author: " << endl;
144   cin >> _author;
145
146   delete[] author;
147   author = new char[strlen(_author) + 1];
148   strcpy(author, _author);
149 }
150
151 void Document::modifyDay()
152 {
153   char * _day = new char[100];
154   cout << "Enter the new date: " << endl;
155   cin >> _day;
156
157   delete[] day;
```

```
158   day = new char[strlen(_day) + 1];
159   strcpy(day, _day);
160 }
161
162 void Document::modifyPages()
163 {
164   int _pages = 0;
165   cout << "Enter the new number of pages:" << endl;
166   cin >> _pages;
167
168   pages = _pages;
169 }
170
171 void Document::modifylastChange()
172 {
173   char * _lastChange = new char[100];
174   cout << "Enter the new day of the lastChange: " << endl;
175   cin >> _lastChange;
176
177   delete[] lastChange;
178   lastChange = new char[strlen(_lastChange) + 1];
179   strcpy(lastChange, _lastChange);
180 }
181
182
183 void Document::modifyData()
184 {
185   enum choice {Title = 1, Subject = 2, Author = 3, Day = 4,
      Pages = 5, LastChange = 6};
186   int nr;
187
188   cout << "\n\nWhat do you want to change? Enter the number
      according to your choice:\n" <<
189     "1 - Title\n" << "2 - Subject\n" << "3 - Author\n" << "4-
      Day\n" <<
190     "5 - Pages\n" << "6 - Last change" << endl;
191   cin >> nr;
192
193   switch (nr)
194   {
195   case 1:
196     modifyTitle();
197     break;
198   case 2:
199     modifySubject();
```

```
200       break;
201     case 3:
202       modifyAuthor();
203       break;
204     case 4:
205       modifyDay();
206       break;
207     case 5:
208       modifyPages();
209       break;
210     case 6:
211       modifylastChange();
212       break;
213     default:
214       cout << "The number entered is not correct...Try again";
215       modifyData();
216       break;
217     }
218 }
219
220 Document::~Document()
221 {
222   cout << "Deleted";
223   delete[] title;
224   delete[] subject;
225   delete[] author;
226   delete[] day;
227   delete[] lastChange;
228   pages = 0;
229 }
```

main.cpp

```
1  #include <iostream>
2  #include "document.h"
3
4  using namespace std;
5
6  int main()
7  {
8    Document * doc = new Document;
9    doc->setAuthor("Andrei");
10   doc->setDay("Marti");
11   doc->setTitle("License");
12   doc->setLastChange("29/09/2017");
```

```
13    doc->setNumberOfPages(100);
14    doc->setSubject("criminal");
15
16    doc->displayDocumentInformation();
17    doc->modifyData();
18    doc->displayDocumentInformation();
19
20    return 0;
21 }
```

Data analysis for second problem

For error handling I used assert()

1)Default Constructor:
Matrix::Matrix(): columns(1), rows(1)
I used the initialization list in order to initialize member's values.
2)Constructor with one parameter:
$Matrix :: Matrix(int_rows) : columns(rows)$
It assign to the member named "row" of the class value of the parameter, and also with this value we initialize the member named "column", so we can create a square matrix.
3)Constructor with two parameters:
$Matrix :: Matrix(int\_rows, int\_columns)$
Assign to the corresponding members, values saved in those parameters.
4)Set value function:
$voidMatrix :: SetValue(introw\_index, intcolumn\_index, intvalue)$
It saves the number saved in "value" parameter to our matrix component from the row: row_index and column: $column\_index$.
5)Get value function:
$intMatrix :: GetValue(introw\_index, intcolumn\_index)const$
Returns matrix component from the corresponding subscripts.
6)Mathematical functions:
$MatrixMatrix :: operator + (constMatrix\&mat)$
$Matrix\&Matrix :: operator = (constMatrix\&mat)$
$MatrixMatrix :: operator - (constMatrix\&mat)$
$MatrixMatrix :: operator * (constMatrix\&mat)$
$MatrixMatrix :: operator * (constint\&nr)$

Mathematical functions use operator overloading. Return type is Matrix, because they return the matrix in which we saved the result after we performed the necesarry operation.

7)Destructor:
Matrix:: Matrix()
It iterates through all rows and deletes all matrix components from those rowas, after which the matrix is deleted.

matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

class Matrix
{
private:
    int ** matrix;
    int columns;
    int rows;

public:
    Matrix();
    Matrix (int);
    Matrix (int, int);
    void SetValue(int, int, int);
    int GetValue(int, int) const;
    Matrix& operator = (const Matrix&);
    Matrix operator + (const  Matrix&);
    Matrix operator - (const Matrix&);
    Matrix operator * (const Matrix&);
    Matrix operator * (const int&);
    void displayMatrix();
    ~Matrix();

};

#endif
```

matrix.cpp

```
#include <iostream>
#include <assert.h>
```

```
3  #include "matrix.h"
4
5  using namespace std;
6
7  Matrix::Matrix(): columns(1), rows(1)
8  {
9      matrix = new int*[rows];
10     for(int i = 0; i < rows; i++)
11         matrix[i] = new int [columns];
12  }
13
14 Matrix::Matrix(int _rows) : columns(_rows)
15 {
16     rows = _rows;
17     matrix = new int*[rows];
18     for (int i = 0; i < _rows; i++)
19             matrix[i] = new int [columns];
20 }
21
22 Matrix::Matrix(int _rows, int _columns)
23 {
24     rows = _rows;
25     columns = _columns;
26
27     matrix = new int*[rows];
28     for (int i = 0; i < rows; i++)
29              matrix[i] = new int [columns];
30 }
31
32 void Matrix::SetValue(int row_index, int column_index, int value
       )
33 {
34     matrix[row_index][column_index] = value;
35 }
36
37 int Matrix::GetValue(int row_index, int column_index) const
38 {
39     int temp = matrix[row_index][column_index];
40     return temp;
41 }
42
43
44 Matrix Matrix::operator + (const Matrix &mat)
45 {
46     int nr_rows = rows;
```

```
47      int nr_columns = columns;
48
49      assert(nr_rows == mat.rows);
50      assert(nr_columns == mat.columns);
51
52      Matrix temp(rows, columns);
53      for(int i = 0; i < nr_rows; i++)
54      {
55          for(int j = 0; j < nr_columns; j++)
56          {
57              temp.matrix[i][j] = matrix[i][j] + mat.matrix[i][j];
58          }
59      }
60      return temp;
61  }
62
63  Matrix& Matrix::operator = (const Matrix &mat)
64  {
65      for(int i = 0; i < rows; i++)
66      {
67        for(int j = 0; j < columns; j++)
68            matrix[i][j] = mat.matrix[i][j];
69      }
70      return *this;
71  }
72  Matrix Matrix::operator - (const Matrix &mat)
73  {
74      int nr_rows = rows;
75      int nr_columns = columns;
76
77      assert(nr_rows == mat.rows);
78      assert(nr_columns == mat.columns);
79
80      Matrix temp(rows, columns);
81      for(int i = 0; i < nr_rows; i++)
82      {
83          for(int j = 0; j < nr_columns; j++)
84              temp.matrix[i][j] = matrix[i][j] - mat.matrix[i][j];
85      }
86      return temp;
87  }
88
89  Matrix Matrix::operator * (const Matrix &mat)
90  {
91      int nr_rows = rows;
```

```
92      int nr_columns = columns;
93
94      assert(nr_rows == mat.rows);
95      assert(nr_columns == mat.columns);
96
97      Matrix temp(rows, columns);
98      for(int i = 0; i < nr_rows; i++)
99      {
100         for(int j = 0; j < nr_columns; j++)
101         {
102         int sum = 0;
103             for(int k = 0; k < nr_rows; k ++)
104             {
105                 sum = sum + matrix[i][k] * mat.matrix[k][j];
106             }
107         temp.matrix[i][j] = sum;
108         }
109     }
110     return temp;
111 }
112
113 Matrix Matrix:: operator * (const int &nr)
114 {
115     Matrix temp(rows, columns);
116     for(int i = 0; i < rows; i++)
117     {
118         for(int j = 0; j < columns; j++)
119             temp.matrix[i][j] = matrix[i][j] * nr;
120     }
121     return temp;
122 }
123
124 void Matrix::displayMatrix()
125 {
126     for (int i = 0; i < rows; i++)
127     {
128         for(int j = 0; j < columns; j++)
129         {
130             cout << matrix[i][j] << "     ";
131         }
132         cout <<"\n";
133     }
134 }
135
136 Matrix::~Matrix()
```

```
{
    for(int i = 0; i < rows; i++)
        delete[] matrix[i];
    delete[] matrix;
    columns = 0;
    rows = 0;
}
```

main.cpp

```
#include <iostream>
#include "matrix.h"

using namespace std;

int main()
{
    Matrix mat1(2, 2);
    Matrix mat2 (2);
    Matrix addresult(2, 2), subresult(2, 2), multresult(2, 2),
    scmultresult(2, 2);

    int elem;

    cout << "\nMatrix 1: " << endl;
    for(int i = 0; i < 2; i++)
    {
        for(int j = 0; j < 2; j++)
        {
            cout<< "Element for " << i + 1 << " row and " << j +
    1 <<" column:" << endl;
            cin >> elem;
            mat1.SetValue(i, j, elem);
        }
    }

    cout << "\nMatrix 2:" << endl;
    for(int i = 0; i < 2; i++)
    {
        for(int j = 0; j < 2; j++)
        {
            cout<< "Element for " << i + 1 << " row and " << j +
    1 <<" column:" << endl;
            cin >> elem;
            mat2.SetValue(i, j, elem);
```

```
33          }
34      }
35      cout << "\nMatrix1: "<< endl;
36      mat1.displayMatrix();
37      cout << "\nMatrix1: "<< endl;
38      mat2.displayMatrix();
39
40      addresult = mat1 + mat2;
41      cout << "\n Addition result" << endl;
42      addresult.displayMatrix();
43
44      subresult = mat1 - mat2;
45      cout << "\n Substraction result" << endl;
46      subresult.displayMatrix();
47
48      multresult = mat1 * mat2;
49      cout << "\n Multiplication result" << endl;
50      multresult.displayMatrix();
51
52
53      scmultresult = mat1 * 3;
54      cout << "\n Scalar multiplication result" << endl;
55      scmultresult.displayMatrix();
56
57      cout << mat1.GetValue(0, 0)<< endl;
58      return 0;
59  }
```

Conclusion:

If we compare classes with structs, we can notice that classes have more advantages, since they are composed of members and methods, so we can create functions that performs some operation with this class without using an additional parameter(in case of structs: pointer to a structure), so we understand one of the basic concepts of Object Oriented Programming paradigm, named encapsulation.

Bibliography

- Handnotes on OOP lesson of Lector dr., conf. univ. M.Kulev. for the FAF – 161 group Chișinău: UTM, 2017.
- http://www.cplusplus.com/doc/tutorial/classes/
- http://www.geeksforgeeks.org/constructors-c/
- http://www.geeksforgeeks.org/destructors-c/