Отчет по Лабораторной работе №2

Технология программирования

Бекауов Артур Тимурович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью работы является написание программы на c++, содержащей описание классов vect и matr, конструкторы и деструктор для каждого класса, набор оператор-функций для операций векторной алгебры и функцию main, использующую вышеописанный инструментарий.

2 Ход лабораторной работы

Описание классов

Программа, написанная мной задаёт классы vect и matr. Экземплярами vect (Рис. 2.1), будут векторы, описываемые четырьмя полями - размерность int dim (целое число), значение double*v (указатель на одномерный массив), номер вектора int num (целое число), и статической переменной static int count, которая ведёт общий счёт созданных векторов.

Затем описаны конструкторы и деструктор:

-vect(); - конструктор по умолчанию (создаёт единичный вектор размера 3).

-vect(vect&x); - конструктор копирования.

-vect(int n); - конструктор создания нулевого вектора размера n.

-vect(int n, double*x); - конструктор создания вектора с заданными значениями

-~vect(); - деструктор вектора

После этого перечисленны методы класса vect и дружественные функции:

-void print(); - функция вывода вектора.

-vect operator+(vect r); - функция сложения векторов (бинарная).

-friend vect operator-(vect&l,vect&r); - друж. функция вычитания векторов (бинарная).

-vect operator-(); - функция вычитания вектора (унарная).

-vect operator=(const vect&r); - функция присвоения вектору значения другого вектора.

-double operator*(vect&r); - функция скалярного умножения векторов.

-friend vect operator*(double k, vect&r); - друж. функция умножения вектора на

число k.

Далее описан дружественный класс matr. Сделанно это, потому что в классе matr появится метод, которому нужен будет доступ к внутренним данным класса vect (а именно к значениям вектора).

Сразу после описания класса я определил и описал статическую переменную count = 0, для её корректного функционирования внутри main.

```
10  class vect
11  {
12      int dim;
13      double*v;
14
15      public:
16      int num;
17      static int count;
18
19      vect();
20      vect(vect&x);
21      vect(int n);
22      vect(int n, double*x);
23      ~vect();
24      void print();
25
26      vect operator+(vect r);
27      friend vect operator-(vect&l,vect&r);
28      vect operator-();
29      vect operator=(const vect&r);
30      double operator*(vect&r);
31      friend vect operator*(double k, vect&r);
32      friend class matr;
33  };
34
35  int vect::count=0;
36
```

Рис. 2.1: Описание класса vect

Экземплярами matr(Рис. 2.2), будут квадратные матрицы, описываемые четырьмя полями (хотя можно обойтись и первыми двумя) - размерность int dim (целое число), значение double**v (указатель на массив указателей на массивы вещ. чисел - т.е двумерный массив вещ. чисел), номер матрицы int num (целое

число), и статической переменной static int count, которая ведёт общий счёт созданных матриц.

Затем описаны конструкторы:

-matr(); - конструктор по умолчанию (создаёт единичную матрицу 3x3).

-matr(matr&x); - конструктор копирования.

-matr(int n); - конструктор создания нулевой матрицы размера nxn.

-matr(int n, double**x); - конструктор создания матрицы nxn с заданными компонентами.

-~matr(); - деструктор матрицы.

После этого перечисленны методы класса matr и дружественные функции:

-void print(); - функция вывода матрицы.

-matr operator+(matr r); - функция сложения матриц.

-matr operator-(matr&r); - функция вычитания матриц (бинарная).

-matr operator-(); - функция вычитания матрицы (унарная).

-matr operator=(const matr r); - функция присвоения матрице значения другой матрицы.

-matr operator*(matr&r); - функция перемножения матриц.

-friend matr operator*(double k, matr&r); - друж. функция умножения матрицы на число.

-vect operator*(vect&r); - функция умножения матрицы на вектор столбец (результатом будет вектор).

Затем я, конечно, определил и описал статическую переменную count = 0.

```
41  class matr
42  {
43      int dim;
44      double**v;
45
46      public:
47      int num;
48      static int count;
49
50      matr();
51      matr(matr&x);
52      matr(int n);
53      matr(int n, double**x);
54      ~matr();
55      void print();
56
57      matr operator+(matr r);
58      matr operator-(matr&r);
59      matr operator-();
60      matr operator=(const matr r);
61      matr operator*(matr&r);
62      friend matr operator*(double k, matr&r);
63      vect operator*(vect&r);
64  };
65
66  int matr::count=0;
```

Рис. 2.2: Описание класса matr

Описание конструкторов, деструктора и методов класс vect

Описание я начал с конструктора вектора по умолчанию (Рис. 2.3). Так как это конструктор, то первым делом я увеличил значение count на 1 (потому что всякий раз, когда вызывается конструктор - создаётся новый вектор). Присвоил парамеру num создаваемого новое значение count. Таким образом созданный

вектор получил свой уникальный номер, по которому я смогу к нему обращаться в описаниях действий конструкторов, деструкторов и методов класса vect (в том числе и при выводе вектора). Затем параметру dim задаю значение 3. А параметру v с помощью оператора new задаю значение указателя на динамическую память в которой находится массив размером в dim вещественных чисел. Далее с помощью цикла for заполняю массив единицами. В конце вывожу сообщение, что конструктор по умолчанию создал вектор №num. Таким образом, конструктор по умолчанию создал единичный вектор размерностью 3. (Как пример - v1(1,1,1)).

```
71  //Конструктор вектора по умолчанию
72  vect::vect()
73  {
74      count ++;
75      num = count;
76      dim = 3;
77      v = new double[dim];
78      for (int i = 0; i < dim; i++)
79      {
80          v[i]=1;
81      }
82      cout<<"Конструктор vect(), создал вектор №"<<num<<endl;
83  }
```

Рис. 2.3: vect: Конструктор по умолчанию

После - я описал конструктор копирования вектора (Рис. 2.4), который получает на вход ссылку на объект класса vect - r. Увеличиваю значение count на 1 и присваиваю его параметру num. Затем пареметру dim присваиваю значение dim вектора r (r.dim). Выделяем динамическую память под массив v, и каждому из значений массива v[i] присваиваю соответствующее значение r.v[i]. В конце вывожу сообщение, что конструктор копирования создал вектор №num. Таким образом, конструктор по умолчанию создал вектор-копию r.

```
85  //Конструктор копирования вектора
86  vect::vect(vect &r)
87  {
88      count ++;
89      num = count;
90      dim = r.dim;
91      v = new double[dim];
92      for (int i=0; i<dim; i++)
93          v[i]=r.v[i];
94      cout<<"Конструктор vect(vect) создал вектор №"<<num<<endl;
95  }
```

Рис. 2.4: vect: Конструктор копирования

Далее я описал конструктор создания нулевого вектора (Рис. 2.5), который получает на вход целое число n - размер будущего вектора. Увеличиваю значение count на 1 и присваиваю его параметру num. Затем параметру dim присваиваю значение n. Выделяем динамическую память под массив v, и каждому из значений массива v[i] присваиваю значение 0. В конце вывожу сообщение, что конструктор создания нулевого вектора создал вектор №num. Таким образом, конструктор создания нулевого вектора создал нулевой вектор размерностью n.

```
98  //Конструктор создания нулевого вектора
99  vect::vect(int n)
100 {
101     count ++;
102     num = count;
103     dim = n;
104     v = new double[dim];
105     for (int i = 0; i < dim; i++)
106         v[i] = 0;
107     cout<<"Конструктор vect(int) создал вектор №"<<num<<endl;
108 }
```

Рис. 2.5: vect: Конструктор создания нулевого вектора

После - я описал конструктор создания вектора с компонентами (Рис. 2.6), который полуает на вход целое число n - размер будущего вектора и указатель на массив вещественных числе *x - компоненты будущего вектора. Увеличиваю

значение count на 1 и присваиваю его параметру num. Затем пареметру dim присваиваю значение n. Выделяем динамическую память под массив v, и каждому из значений массива v[i] присваиваю значение x[i]. В конце вывожу сообщение, что конструктор создания вектора с компонентами создал вектор №num. Таким образом, конструктор создания вектора с компонентами создал вектор размерностью n, с компонентами массива x.

```
110  //Конструктор создания вектора с компонентами;
111  vect::vect(int n, double*x)
112  {
113      count ++;
114      num = count;
115      dim = n;
116      v = new double[dim];
117      for (int i = 0; i < dim; i++)
118          v[i] = x[i];
119      cout<<"Конструктор vect(int, double) создал вектор №"<<num<<endl;
120  }
```

Рис. 2.6: vect: Конструктор создания вектора с компонентами

Далее я описал деструктор вектора (Рис. 2.7). С помощью оператора delete освобождаю динамическую память, на которую указывет указатель v. Параметру dim присваиваю значеник 0. В конце вывожу сообщение, что деструктор вектора ликвидировал вектор №num. Таким образом деструктор вектора ликвидировал вектор №num и освободил соотствующую ему динамическую память.

```
122  //Деструктор вектора
123  vect::~vect()
124  {
125      delete v;
126      dim = 0;
127      cout<<"Деструктор ~vect() ликвидировал вектор №"<<num<<endl;
128  }
```

Рис. 2.7: vect: Деструктор вектора

Затем я описал функцию вывода вектора (Рис. 2.8). Вывожу сообщение, указы-

вающее номер выводимого вектора и его размерность. Потом с помощью цикла for вывожу каждое из значений компонент вектора.

```
130  // Вывод вектора
131  void vect::print()
132  {
133      cout<<"Вектор №"<<num<<" размерностью "<<dim<<" имеет значения:"<<endl;
134      for (int i = 0; i < dim; i++)
135      {
136          cout<<"dim "<<i+1<<" = "<<v[i]<<endl;
137      }
138      cout<<endl;
139  }
```

Рис. 2.8: vect: Функция print

После - я описал функция сложения векторов (Рис. 2.9), которая получает на вход вектор r (Функция бинарная, первый (левый вектор) - тот, к которму применяется данный метод). С помощью конструктора создания нулевого вектора создаю вектор tmp размерностью dim. Далее в цикле for меняю значения массива tmp.v[i]=0 на сумму v[i]+r.v[i]. В конце вывожу сообщение, что создан вектор со значением суммы векторов №num + №r.num. (“Создан” я написал потому, что вектор tmp внутри данного метода действительно создаётся конструктором копирования). Возвращается вектор tmp (Результатом работы метода является объект класса vect).

```
141  // Функция сложения векторов
142  vect vect::operator+(vect r)
143  {
144      vect tmp(dim);
145      for (int i = 0; i < dim; i++)
146      {
147          tmp.v[i]=v[i]+r.v[i];
148      }
149      cout<<"Создан вектор со значением = v"<<num<<" + v"<<r.num<<endl;
150      return tmp;
151  }
```

Рис. 2.9: vect: Функция сложения векторов

Затем я описал функцию бинарного вычитания векторов (Рис. 2.10), которая получает на вход ссылки на уменьшаемый вектор l и вычитаемый вектор r. С помощью конструктора создания нулевого вектора создаю вектор tmp размерностью l.dim. Далее в цикле for меняю значения массива tmp.v[i]=0 на разность l.v[i]-r.v[i]. В конце вывожу сообщение, что создан вектор со значением разности векторов №l.num - №r.num.Возвращается вектор tmp.

```
153  // Функция вычитания векторов бинарная
154  vect operator-(vect&l, vect&r)
155  {
156      vect tmp(l);
157      for (int i = 0; i < tmp.dim; i++)
158      {
159          tmp.v[i]=l.v[i]-r.v[i];
160      }
161      cout<<"Создан вектор со значением = v"<<l.num<<" - v"<<r.num<<endl;
162      return tmp;
163  }
```

Рис. 2.10: vect: Функция вычитания векторов бинарная

Потом я описал функцию унарного вычитания вектора (Рис. 2.11). С помощью конструктора создания нулевого вектора создаю вектор tmp размерностью dim. Далее в цикле for меняю значения массива tmp.v[i]=0 на -1 * v[i]. В конце вывожу сообщение, что создан вектор со значением -1 * №num.Возвращается вектор tmp.

```
165  //Функция вычитания векторов унарная
166  vect vect::operator-()
167  {
168      vect tmp(dim);
169      for (int i = 0; i < dim; i++)
170      {
171          tmp.v[i]=-1*v[i];
172      }
173      cout<<"Создан вектор со значением = -v"<<num<<endl;
174      return tmp;
175  }
```

Рис. 2.11: vect: Функция вычитания вектора унарная

После этого я описал функцию присвоения вектору значения другого вектора (Рис. 2.12), которая получает на вход ссылку на const вектор r. Параметру dim вектора, к которому применён метод, задаём значение r.dim. Затем с помощью цикла for значениям v[i] присваиваем r.v[i]. В конце вывожу сообщение, что вектору num присвоено значение вектора r.num. Возвращается вектор, к которому применён метод (с помощью указателя this).

```
177  // Функция присвоения вектору значения другого вектора
178  vect vect::operator=(const vect&r)
179  {
180      dim = r.dim;
181      for (int i = 0; i < dim; i++)
182      {
183          v[i]=r.v[i];
184      }
185      cout<<"Вектору v"<<num<<" присвоено значение вектора v"<<r.num<<endl;
186      return *this;
187  }
```

Рис. 2.12: vect: Функция присвоения

Далее я описал функцию скалярного умножения векторов (Рис. 2.13), которая получает на вход ссылку на вектор r. Объявляю и описываю вещественное число tmp = 0. Потом с помощью цикла for, суммирую в tmp все произведения вида v[i]*r.v[i]. В конце вывожу сообщение, что создана скалярная переменная со значением №num №r.num. Возвращаю значение tmp.

```
189  // Функция скалярного умножения векторов;
190  double vect::operator*(vect&r)
191  {
192      double tmp = 0;
193      for (int i = 0; i < dim; i++)
194      {
195          tmp+=v[i]*r.v[i];
196      }
197      cout<<"Создана переменная со скалярным значением = v"<<num<<" * v"<<r.num<<endl;
198      return tmp;
199  }
```

Рис. 2.13: vect: Функция присвоения

Затем я описал функцию умножения вектора на число (Рис. 2.14), которая получает на вход вещественное число k, и ссылку на вектор r. С помощью конструктора создания нулевого вектора создаю вектор tmp размерностью r.dim. Далее в цикле for меняю значения массива tmp.v[i]=0 на k * r.v[i]. В конце вывожу сообщение, что создан вектор со значением k * №num.Возвращается вектор tmp.

```
201  // Функция умножения вектора на число
202  vect operator*(double k, vect&r)
203  {
204      vect tmp(r);
205      for (int i = 0; i < tmp.dim; i++)
206      {
207          tmp.v[i]=r.v[i]*k;
208      }
209      cout<<"Создан вектор со значением = "<<k<<" * v"<<r.num<<endl;
210      return tmp;
211  }
```

Рис. 2.14: vect: Функция умножения вектора на число

Описание конструкторов, деструктора и методов класс matr

Описание я начал с конструктора матрицы по умолчанию (Рис. 2.15). Увеличиваю значение count на 1 и присваиваю его параметру num. Затем пареметру dim присваиваю значение 3 (таким образом матрица будет 3x3). А параметру v с помощью оператора new задаю значение указателя на динамическую память в которой находится массив размером в dim из указателей на массивы вещественных чисел - таким образом v - указатель на двумерный массив вещественных чисел. Далее с помощью двух циклов for заполняю массив единицами по основной диагонали (где i=j, т.е номер строки равен номеру ряда) и нулями на всех остальных позициях. Таким образом, конструктор по умолчанию создал единичную матрицу размерностью 3x3.

```
215  //Конструктор матрицы по умолчанию (единичной) --- Работает только в форме matr m1 = matr();
216  matr::matr()
217  {
218      count ++;
219      num = count;
220      dim = 3;
221      v = new double*[dim];
222      for (int i = 0; i < dim; i++)
223      {
224          v[i] = new double [dim];
225          for (int j = 0; j < dim; j++)
226          {
227              if (i==j)
228              {
229                  v[i][j]=1;
230              }
231              else
232              {
233                  v[i][j]=0;
234              }
235          }
236
237      }
238      //cout<<"Конструктор matr(), создал матрицу №"<<num<<endl;
239  }
```

Рис. 2.15: matr: Конструктор по умолчанию

После - я описал конструктор копирования матрицы (Рис. 2.16), который получает на вход ссылку на объект класса matr - r. Увеличиваю значение count на 1 и присваиваю его параметру num. Затем параметру dim присваиваю значение dim матрицы r (r.dim). Выделяем динамическую память под двумерный массив v, и каждому из значений массива v[i][j] присваиваю соответствующее значение r.v[i][j]. Таким образом, конструктор по умолчанию создал матрицу-копию r.

```
241  //Конструктор копирования матрицы
242  matr::matr(matr &r)
243  {
244      count ++;
245      num = count;
246      //cout<<"Конструктор matr(matr) создал матрицу №"<<num<<endl;
247      dim = r.dim;
248      v = new double*[dim];
249      for (int i = 0; i < dim; i++)
250      {
251          v[i] = new double [dim];
252          for (int j = 0; j < dim; j++)
253          {
254              v[i][j]=r.v[i][j];
255          }
256      }
257  }
258
```

Рис. 2.16: matr: Конструктор копирования

Далее я описал конструктор создания нулевой матрицы (Рис. 2.17), который получает на вход целое число n - размер будущей матрицы. Увеличиваю значение count на 1 и присваиваю его параметру num. Затем пареметру dim присваиваю значение n. Выделяем динамическую память под двумерный массив v, и каждому из значений массива v[i][j] присваиваю значение 0. Таким образом, конструктор создания нулевой матрицы создал нулевую матрицу размерностью nxn.

```
260  //Конструктор создания нулевой матрицы, размера n.
261  matr::matr(int n)
262  {
263      count ++;
264      num = count;
265      //cout<<"Конструктор matr(int) создал матрицу №"<<num<<endl;
266      dim = n;
267      v = new double*[dim];
268      for (int i = 0; i < dim; i++)
269      {
270          v[i] = new double [dim];
271          for (int j = 0; j < dim; j++)
272          {
273              v[i][j]=0;
274          }
275      }
276  }
277
```

Рис. 2.17: matr: Конструктор создания нулевой матрицы

После - я описал конструктор создания матрицы с компонентами (Рис. 2.18),

который полуает на вход целое число n - размер будущей матрицы и указатель на двумерный массив вещественных числе **x - компоненты будущей матрицы. Увеличиваю значение count на 1 и присваиваю его параметру num. Затем пареметру dim присваиваю значение n. Выделяем динамическую память под двумерный массив v, и каждому из значений массива v[i][j] присваиваю значение x[i][j]. Таким образом, конструктор создания матрицы с компонентами создал матрицу размерностью nxn, с компонентами двумерного массива x.

```
278  //Конструктор создания матрицы с компонентами x;
279  matr::matr(int n, double**x)
280  {
281      count ++;
282      num = count;
283      //cout<<"Конструктор matr(int, double) создал матрицу №"<<num<<endl;
284      dim = n;
285      v = new double*[dim];
286      for (int i = 0; i < dim; i++)
287      {
288          v[i] = new double [dim];
289          for (int j = 0; j < dim; j++)
290          {
291              v[i][j]=x[i][j];
292          }
293      }
294  }
295
```

Рис. 2.18: matr: Конструктор создания матрицы с компонентами

Далее я описал деструктор матрицы (Рис. 2.19). С помощью цикла for и оператора delete освобождаю динамическую память, на которую указывет указатель v. Таким образом деструктор вектора ликвидировал матрицу №num и освободил соотствующую ей динамическую память.

```
296  //Деструктор матрицы
297  matr::~matr()
298  {
299      //cout<<"Деструктор ~matr() ликвидировал матрицу №"<<num<<endl;
300      for (int i = 0; i < dim; i++)
301      {
302          delete [] v[i];
303      }
304  }
305
```

Рис. 2.19: matr: Деструктор матрицы

Затем я описал функцию вывода матрицы (Рис. 2.20). Вывожу сообщение, указывающее номер выводимой матрицы и её размерность. Потом с помощью двух циклов for вывожу каждое из значений компонент матрицы.

```
306  // Вывод матрицы
307  void matr::print()
308  {
309      cout<<"Матрица №"<<num<<" размерностью "<<dim<<'x'<<dim<<" имеет значения:"<<endl; //тут 
310      for (int i = 0; i < dim; i++)
311      {
312          for (int j = 0; j < dim; j++)
313          {
314              cout<<setw(4)<<v[i][j]<<"  ";
315          }
316          cout<<endl;
317      }
318      cout<<endl;
319  }
320
```

Рис. 2.20: matr: Функция print

После - я описал функцию сложения матриц (Рис. 2.21), которая получает на вход матрицу r. С помощью конструктора создания нулевой матрицы создаю матрицу tmp размерностью dimxdim. Далее в цикле for меняю значения массива tmp.v[i][j]=0 на сумму v[i][j]+r.v[i][j]. Возвращается матрица tmp (Результатом работы метода является объект класса matr).

```
321  // Функция сложения матриц
322  matr matr::operator+(matr r)
323  {
324      matr tmp(dim);
325      for (int i = 0; i < dim; i++)
326      {
327          for (int j = 0; j < dim; j++)
328          {
329              tmp.v[i][j]=v[i][j]+r.v[i][j];
330          }
331      }
332      return tmp;
333  }
```

Рис. 2.21: matr: Функция сложения матриц

Затем я описал функцию бинарного вычитания матриц (Рис. 2.22), которая получает на вход ссылку на вычитаемую матрицу r. С помощью конструктора

создания нулевой матрицы создаю матрицу tmp размерностью dimxdim. Далее в циклах for меняю значения массива tmp.v[i][j]=0 на разность v[i][j]-r.v[i][j]. Возвращается матрица tmp.

```
335  // Функция вычитания матриц бинарная
336  matr matr::operator-(matr&r)
337  {
338      matr tmp(dim);
339       for (int i = 0; i < dim; i++)
340      {
341          for (int j = 0; j < dim; j++)
342          {
343              tmp.v[i][j]=v[i][j]-r.v[i][j];
344          }
345      }
346      return tmp;
347  }
```

Рис. 2.22: matr: Функция вычитания матриц бинарная

Потом я описал функцию унарного вычитания матрицы (Рис. 2.23). С помощью конструктора создания нулевой матрицы создаю матрицу tmp размерностью dimxdim. Далее в циклах for меняю значения массива tmp.v[i]=0 на -1 * v[i]. Возвращается матрица tmp.

```
349  //Функция вычитания матриц унарная
350  matr matr::operator-()
351  {
352      matr tmp(dim);
353      for (int i = 0; i < dim; i++)
354      {
355          for (int j = 0; j < dim; j++)
356          {
357              tmp.v[i][j]=-v[i][j];
358          }
359      }
360      return tmp;
361  }
```

Рис. 2.23: matr: Функция вычитания матрицы унарная

После этого я описал функцию присвоения матрице значения другой матрицы (Рис. 2.24), которая получает на вход ссылку на const матрицу r. Параметру dim матрицы, к которому применён метод, задаём значение r.dim. Затем с помощью циклов for значениям v[i][j] присваиваем r.v[i][j]. Возвращается матрица, к

которой применён метод (с помощью указателя this).

```
363  // Функция присвоения матрице значения другой матрицы
364  matr matr::operator=(const matr r)
365  {
366      dim = r.dim;
367      for (int i = 0; i < dim; i++)
368      {
369          for (int j = 0; j < dim; j++)
370          {
371              v[i][j]=r.v[i][j];
372          }
373      }
374      return *this;
375  }
```

Рис. 2.24: matr: Функция присвоения

Далее я описал функцию перемножения матриц (Рис. 2.25), которая получает на вход ссылку на матрицу r. С помощью конструктора создания нулевой матрицы создаю матрицу tmp размерностью dimxdim. Далее с помощью трёх циклов for в каждом значении tmp.v[i][j] суммируем значения вида v[i][k]*r.v[k][j] (где k принадлежит [0;dim-1]). Возвращается матрица tmp.

```
// Функция перемножения матриц;
 matr matr::operator*(matr&r)
{
    matr tmp(dim);
    for (int i = 0; i < dim; i++)
    {
        for (int j = 0; j < dim; j++)
        {
            for (int k = 0; k < dim; k++)
            {
                tmp.v[i][j]+=v[i][k]*r.v[k][j];
            }
        }
    }
    return tmp;
}
```

Рис. 2.25: matr: Функция присвоения

Затем я описал функцию умножения матрицы на число (Рис. 2.26), которая

получает на вход вещественное число k, и ссылку на матрицу r. С помощью конструктора создания нулевой матрицы создаю матрицу tmp размерностью r.dimxr.dim. Далее в циклах for меняю значения массива tmp.v[i][j]=0 на k * r.v[i][j]. Возвращается матрица tmp.

```
// Функция умножения матрицы на число
matr operator*(double k, matr&r)
{
    matr tmp(r.dim);
    for (int i = 0; i < tmp.dim; i++)
    {
        for (int j = 0; j < tmp.dim; j++)
        {
            tmp.v[i][j]=r.v[i][j]*k;
        }
    }
    return tmp;
}
```

Рис. 2.26: matr: Функция умножения матрицы на число

Затем я описал функцию умножения матрицы на вектор-столбец (Рис. 2.27), которая получает на вход ссылку на вектор r. С помощью конструктора создания нулевого вектора создаю вектор tmp размерностью dim. Далее в циклах for меняю значения вектора tmp.v[i]=0 на сумму всех элементов вида v[i][j]*r.v[i] (где j принадлежит [0;dim-1]). Возвращается вектор tmp.

```
// Функция умножения матрицы на вектор-столбец
vect matr::operator*(vect&r)
{
    vect tmp(dim);
    for (int i = 0; i < dim; i++)
    {
        for (int j = 0; j < dim; j++)
        {
            tmp.v[i]+=v[i][j]*r.v[i];
        }
    }
    return tmp;
}
```

Рис. 2.27: matr: Функция умножения матрицы на вектор-столбец

Функция main

В функции main я привёл пример работы каждого конструктора и метода классов vect и matr. Также привёл 2 более сложных примера, показывающих, что работает двойное присваивание и сложные операции. (К примеру matr1 + 3*mart2).

3 Выводы

В ходе лабораторной работы я написал программу на c++, содержащую описание классов vect и matr, конструкторы и деструктор для каждого класса, набор оператор-функций для операций векторной алгебры и функцию main, использующую вышеописанный инструментарий.