

Programming 2 - Assignment 2

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

April 27, 2022

1 Testing the Program

For testing the Program, or in specific, the class, a series of tests was performed by testing different methods of this through different main-methods.

1.1 testing the specified commands

In this section the commands given in the assignment instructions will be tested.

The following code-block shows the methods used to perform the first test. As shown, every operation was performed in two variables, with multiple inclusions of both *add()* methods and the *println()* method.

```
1 #include "DistPoly.h"
2 #include <string>
3
4 string vars[2] = { "x", "y" };
5
6 int main() {
7
8     // some exponent vectors ("power products")
9     int e1[2] = { 1, 2 }; int e2[2] = { 2, 1 }; int e3[2] = { 1, 0 };
10    int e4[2] = { 0, 1 }; int e5[2] = { 0, 0 }; int e6[2] = { 2, 2 };
11
12    // construct zero polynomial in two variables, then add monomials
13    string vars[2] = { "x", "y" };
14    DistPoly p(2, vars);
15    p.add(3, e1).add(5, e2).add(7, e3).add(11, e4).add(13, e5);
16
17    // construct zero polynomial in two variables, then add monomials
18    DistPoly q(2, vars);
19    q.add(11, e4).add(-3, e2).add(2, e6).add(-2, e2);
20
21    // print p and q
22    p.println();
23    q.println();
24
25    // set p to p+2*q and print it
26
27    DistPoly r = p;
28    r.add(q).add(q);
29    p = r;
30    p.println();
31
32    return 0;
33 }
```

This is the output, that was created by the code above.

```
5x^2y+3xy^2+7x+11y+13
2x^2y^2-5x^2y+11y
4x^2y^2-5x^2y+3xy^2+7x+33y+13
```

1.2 testing error messages

In this section, we will test different kinds of errors that can occur during programming with this class. We will try to produce error messages.

1.2.1

2 The Class - DistPoly.h

```

1  #pragma once
2
3  #include<string>
4
5  using namespace std;
6
7  class Monom
8  {
9  public:
10     int coeff;
11     int* exps;
12     int n;
13
14     //constructor
15     Monom(int coeff, int* exps, int n);
16     Monom();
17
18     //copy constructor, copy assignment operator, destructor
19     Monom& operator=(Monom& m);
20
21     //destructor
22     ~Monom();
23
24 };
25
26
27 class DistPoly
28 {
29 private:
30     int n; //number of variables
31     string* vars; //names of the variables
32     Monom* monoms; //pointer to an array of monomials
33     int m; //number of potential monoms in this polynomial (allocated memory)
34     int am; //actual number of monoms in this polynomial -1
35 public:
36
37     //constructor
38     DistPoly(int n, string* vars);
39
40     //copy constructor, copy assignment operator, destructor
41     DistPoly(DistPoly& p);
42     DistPoly& operator=(DistPoly& p);
43     ~DistPoly();
44
45     DistPoly& add(int coeff, int* exps);
46     DistPoly& add(DistPoly& p);
47     //void println_brkts(); //prints the polynomial in brackets-style
48     void println();
49
50     void resize(int factor); //enlarges the polynomial,
51
52     int sort(int* exps, int n, int j); //gives back the position in which the monom with the
        exponents exps should be inserted
53 };

```

3 The Class - DistPoly.cpp

Note: The copy constructor could also be implemented by using the add function.

```

1  #include "DistPoly.h"
2  #include <iostream>
3
4  using namespace std;
5
6  DistPoly& DistPoly::add(int coeff, int* exps) {
7      if (coeff != 0) {
8          for (int j = 0; j <= this->m; j++) {
9
10             int k = 1;
11             if (j < this->m) {
12                 k = sort(exps, this->n, j);
13             }
14             if (k == 0) {
15                 if (this->monoms[j].coeff + coeff == 0) {
16                     for (int l = j; l < this->am+1; l++) { //shifts the monoms into the
17                         gap to fill it
18                         this->monoms[l] = this->monoms[l + 1];
19                     }
20                     this->am--;
21                 }
22                 else {
23                     this->monoms[j].coeff += coeff;
24                 }
25                 break;
26             }
27             else if (k == -1) {
28                 if (this->m == this->am+1) {
29                     this->resize(2);
30                 }
31
32                 for (int l = this->am + 1; l > j; l--) {
33                     this->monoms[l] = this->monoms[l-1];
34                 }
35                 this->monoms[j].coeff = coeff;
36                 this->monoms[j].exps = exps;
37                 this->am++;
38                 break;
39             }
40         }
41     }
42
43     return *this;
44 }
45
46 DistPoly& DistPoly::add(DistPoly& p) {
47     if (this->n != p.n) {
48         cout << "Error: the number of variables of two added polynomials is different";
49     }
50     for (int i = 0; i < this->n; i++) {
51         if (this->vars[i] != p.vars[i]) {
52             cout << "Error: the variables of two added polynomials do not match";
53             exit(1);
54         }

```

```

55     }
56     if (p.am != 0) {
57         for (int i = 0; i < p.am; i++){
58             this->add(p.monoms[i].coeff, p.monoms[i].exps);
59         }
60     }
61
62     return *this;
63 }
64
65 int DistPoly::sort(int* exps, int n, int j) {
66     for (int i = 0; i < n; i++) {
67         if (this->monoms[j].exps[i] > exps[i]) {
68             return 1;
69         }
70         else if (this->monoms[j].exps[i] < exps[i]) {
71             return -1;
72         }
73     }
74     return 0;
75 }
76
77 void DistPoly::println() {
78     if (n == 0 || m==0 || am==0) {
79         cout << "0" << endl;
80     }
81     else {
82         for (int i = 0; i < m; i++) {
83             if (this->monoms[i].coeff != 0) {
84                 cout << this->monoms[i].coeff;
85
86                 for (int j = 0; j < n; j++) {
87                     if (this->monoms[i].exps[j] == 1) {
88                         cout << this->vars[j];
89                     }
90                     else if (this->monoms[i].exps[j] != 0) {
91                         cout << this->vars[j];
92                         cout << "^" << this->monoms[i].exps[j];
93                     }
94                 }
95                 if (i < am && this->monoms[i+1].coeff > 0) {
96                     cout << "+";
97                 }
98             }
99         }
100         cout << "\n";
101     }
102 }
103
104 DistPoly::DistPoly(int n, string* vars) {
105     this->n = n;
106     this->vars = new string[n];
107     for (int i = 0; i < n; i++) {
108         this->vars[i] = vars[i];
109     }
110     this->m = 1;
111     this->am = 0;
112     this->monoms = new Monom[m];
113     for (int j = 0; j < m; j++) {
114         this->monoms[j].coeff = 0;

```

```

115     this->monoms[j].exps = new int[n];
116     this->monoms[j].n = n;
117     for (int i = 0; i < n; i++) {
118         this->monoms[j].exps[i] = 0;
119     }
120 }
121 }
122
123 DistPoly::DistPoly(DistPoly& p) {
124     this->n = p.n;
125     this->vars = new string[n];
126     for (int i = 0; i < n; i++) {
127         this->vars[i] = p.vars[i];
128     }
129     this->m = p.m;
130     this->am = p.am;
131     this->monoms = new Monom[this->m];
132     //check if monomial is null, e.g. is actually a monomial, this check should be
        included everywhere, where such copying actions are performed
133     for (int i = 0; i < m; i++) {
134         this->monoms[i].coeff = p.monoms[i].coeff;
135         this->monoms[i].n = p.n;
136         for (int j = 0; j < p.n; j++) {
137             this->monoms[i].exps[j] = p.monoms[i].exps[j];
138         }
139     }
140 }
141 }
142
143 DistPoly& DistPoly::operator=(DistPoly& p) {
144     this->n = p.n;
145     delete[] vars;
146     this->vars = new string[n];
147     for (int i = 0; i < n; i++) {
148         this->vars[i] = p.vars[i];
149     }
150     this->m = p.m;
151     this->am = p.am;
152     delete[] monoms;
153     this->monoms = new Monom[this->m];
154     for (int i = 0; i < m; i++) {
155         this->monoms[i].coeff = p.monoms[i].coeff;
156         this->monoms[i].n = p.n;
157         for (int j = 0; j < p.n; j++) {
158             this->monoms[i].exps[j] = p.monoms[i].exps[j];
159         }
160     }
161
162     return *this;
163 }
164
165 //problem is because it tries to delete a pointer, that has already been deleted ->
    better: create constructor and destructor for class Monom
166 DistPoly::~DistPoly() {
167     //for (int i = 0; i < this->m; i++) {
168     //     delete[] this->monoms[i].exps;
169     //}
170     delete[] this->vars;
171     delete[] this->monoms;
172 }

```

```

173
174 void DistPoly::resize(int factor) {
175     if (factor > 0) {
176         Monom* NewMonoms = new Monom[(factor * this->m) + 1];
177         for (int i = 0; i < this->m; i++) {
178             //Monom temporary(this->monoms[i].coeff, this->monoms[i].exps, this->n);
179             //NewMonoms[i] = temporary;
180             NewMonoms[i] = this->monoms[i];
181         }
182         //for (int i = this->m; i < (this->m) * factor; i++) { //initializes the remaining
            //elements of the array with the standard value 0
183             // NewMonoms[i].coeff = 0;
184             // NewMonoms[i].exps = new int[this->n];
185             // for (int j = 0; j < this->n; j++) {
186                 // NewMonoms[i].exps[j] = 0;
187             // }
188         //}
189         delete[] this->monoms;
190         //for (int i = 0; i < n; i++) { //this handles what the missing destructor for the
            //Monoms would otherwise do
191             // if (monoms[i].exps != 0) {
192                 // delete[] monoms[i].exps;
193             // }
194         //}
195         this->monoms = NewMonoms;
196         this->m = factor * (this->m) + 1;
197     }
198     else{
199         cout << "Error: factor must be greater than 0";
200     }
201 }
202 //idea: create new array of monoms with emty constructor, then initialize them by
    //assigning a newly constructed monom (with different constructor) to every element of
    //the array
203
204
205 //constructor
206 Monom::Monom(int coeff, int* exps, int n) {
207     this->n = n;
208     this->coeff = coeff;
209     this->exps = new int[n]; //creates a new array of exponents, this is in order to have
        //separate pointers and deallocate their respective memory later (for DistPoly)
210     for (int i = 0; i < n; i++) {
211         this->exps[i] = exps[i];
212     }
213 }
214
215 //
216 //
217 ////constructor
218 Monom::Monom() {
219     this->n = 1;
220     this->coeff = 0;
221     this->exps = new int[n];
222     for (int i = 0; i < this->n; i++) {
223         this->exps[i] = 0;
224     }
225 }
226 //
227 //

```



```
228 ////copy assignment operator
229 Monom& Monom::operator=(Monom& m) {
230     this->n = m.n;
231     this->coeff = m.coeff;
232     this->exps = new int[n]; //creates a new array of exponents, this is in order to have
        seperate pointers and deallocate their respective memory later (for DistPoly)
233     for (int i = 0; i < n; i++) {
234         this->exps[i] = m.exps[i];
235     }
236
237     return *this;
238 }
239
240 //destructor
241 //Monom::~Monom() {
242 //     delete[] exps;
243 //}
```