Programming 2 - Assignment 4

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

May 25, 2022

1 Testing the Program

This time, no tests for the implemented Error messages were recorded because all of them occurred at least once during the creation of the Code.

1.1 Testing the Program - Main

The following code was used to test multiple possible operations allowed by the program.

```
1  //*****************************************************************
2  // "Main.cpp"
3  //
4  // is used to test the functions of Ring and its subclasses.
5  //
6  // created by: Felix Dressler - 24.05.2022
7  //*****************************************************************
8
9  #include "Integer.h"
10 #include "RecPoly.h"
11 #include <iostream>
12
13 using namespace std;
14
15 int main() {
16 //tests from the assignment
17     Ring* c[] = { new Integer(-5), new Integer(2), new Integer(0), new Integer(-3) };
18
19     RecPoly* p = new RecPoly("x", 4, c); // p = -3x^3 + 2x - 5
20     cout << p->str() << endl;
21
22     RecPoly* q = // q = p+p = -6x^3 + 4x - 10
23         dynamic_cast<RecPoly*>(p->operator+(p));
24     cout << q->str() << endl;
25
26     RecPoly* r = // r = p*q = 50 - 40 x + 8 x^2 + 60 x^3 - 24 x^4 + 18 x^6
27         dynamic_cast<RecPoly*>(p->operator*(q));
28     cout << r->str() << endl;
29
30 //additional tests
31
32 //zero
33
34     RecPoly* z = //zero polynomial
35         dynamic_cast<RecPoly*>(p->zero());
36     cout << z->str() << endl;
37
38     RecPoly* z2 = // = p + 0
39         dynamic_cast<RecPoly*>(p->operator+(z));
40     cout << z2->str() << endl;
41
42     RecPoly* z3 = // = p * 0
43         dynamic_cast<RecPoly*>(p->operator*(z));
44     cout << z3->str() << endl;
45
46 //negation
47
48     RecPoly* g = //= -p
49         dynamic_cast<RecPoly*>(p->operator-());
```

```
50      cout << g->str() << endl;
51
52      RecPoly* g2 = // = -p + p = 0
53          dynamic_cast<RecPoly*>(g->operator+(p));
54      cout << g2->str() << endl;
55
56  //multivariate polynomials
57
58      Ring* k[] = { dynamic_cast<Ring*>(p),dynamic_cast<Ring*>(q)};
59      Ring* l[] = { dynamic_cast<Ring*>(p) };
60
61      RecPoly* s = new RecPoly("y", 2, k);
62      cout << s->str() << endl;
63
64      RecPoly* s2 = new RecPoly("y", 1, l);
65      cout << s2->str() << endl;
66
67      RecPoly* s3 = // s2 + s
68          dynamic_cast<RecPoly*>(s2->operator+(s));
69      cout << s3->str() << endl;
70
71      RecPoly* s4 = //s2*s
72          dynamic_cast<RecPoly*>(s2->operator*(s));
73      cout << s4->str() << endl;
74
75      return 0;
76  }
```

1.1.1 Output

The following output was produced with the expected results:

Because this *RecPoly* could be implemented with every kind of *Ring*, negative coefficients were printed as +-a.

```
1       (-5+2*x^1+-3*x^3)
2       (-10+4*x^1+-6*x^3)
3       (50+-40*x^1+8*x^2+60*x^3+-24*x^4+18*x^6)
4       0
5       (-5+2*x^1+-3*x^3)
6       0
7       (5+-2*x^1+3*x^3)
8       0
9       ((-5+2*x^1+-3*x^3)+(-10+4*x^1+-6*x^3)*y^1)
10      ((-5+2*x^1+-3*x^3))
11      ((-10+4*x^1+-6*x^3)+(-10+4*x^1+-6*x^3)*y^1)
12      ((25+-20*x^1+4*x^2+30*x^3+-12*x^4+9*x^6)+(50+-40*x^1+8*x^2+60*x^3+-24*x^4+18*x^6)*y
            ^1)
```

2 The Program

2.1 The Program - Ring

```
1   //*************************************************************
```

```
2  // "Ring.h"
3  //
4  // contains the pure abstract class Ring, which defines the
5  // base-functionality that should be provided by a Ring.
6  //
7  // created by: Felix Dressler - 24.05.2022
8  //*****************************************************************
9
10 #pragma once
11 #include<string>
12
13 using namespace std;
14
15 class Ring {
16 public:
17   // destructor
18   virtual ~Ring() {}
19
20   // a heap-allocated duplicate of this element
21   virtual Ring* clone() = 0;
22
23   // the string representation of this element
24   virtual string str() = 0;
25
26   // the constant of the type of this element and the inverse of this element
27   virtual Ring* zero() = 0;
28   virtual Ring* operator-() = 0;
29
30   // sum and product of this element and c
31   virtual Ring* operator+(Ring* c) = 0;
32   virtual Ring* operator*(Ring* c) = 0;
33
34   // comparison function
35   virtual bool operator==(Ring* c) = 0;
36 };
```

2.2 The Program - Integer

```
1  //*****************************************************************
2  // "Integer.h
3  //
4  // contains the Class Integer, which is derived from the class Ring.
5  // It implements the Ring of integers.
6  //
7  // created by: Felix Dressler - 24.05.2022
8  //*****************************************************************
9
10 #pragma once
11 #include"Ring.h"
12
13 class Integer : public Ring {
14 private:
15   int n;
16 public:
17   // integer with value n (default 0)
18   Integer(int n = 0);
19
20   // destructor
```

```
21    virtual ~Integer() {
22
23    }
24
25    // a heap-allocated duplicate of this element
26    virtual Ring* clone();
27
28    // the string representation of this element
29    virtual string str();
30
31    // the constant of the type of this element and the inverse of this element
32    virtual Ring* zero();
33    virtual Ring* operator-();
34
35    // sum and product of this element and c
36    virtual Ring* operator+(Ring* c);
37    virtual Ring* operator*(Ring* c);
38
39    // comparison function
40    virtual bool operator==(Ring* c);
41 };
```

```
1  //*****************************************************************
2  // "Integer.cpp"
3  //
4  // implements the functionality of the class Integer with its
5  // Ring operations
6  //
7  // created by: Felix Dressler - 24.05.2022
8  //*****************************************************************
9
10 #include"Integer.h"
11 #include<iostream>
12
13 using namespace std;
14
15 // integer with value n (default 0)
16 Integer::Integer(int n) {
17     this->n = n;
18 }
19
20 // a heap-allocated duplicate of this element
21 Ring* Integer::clone() {
22     Integer* c = new Integer(this->n);
23
24     return c;
25 }
26
27 // the string representation of this element
28 string Integer::str() {
29
30     return to_string(this->n);;
31 }
32
33 // the constant of the type of this element and the inverse of this element
34 Ring* Integer::zero() {
35
36     return new Integer(0);
37 }
```

```
38
39 Ring* Integer::operator-() {
40
41     return new Integer(-(this->n));
42 }
43
44 // sum and product of this element and c
45 Ring* Integer::operator+(Ring* c) {
46     Integer* x = dynamic_cast<Integer*>(c);
47
48     //if cast is unsuccessful, we exit
49     if (x == 0) {
50         cout << "Error: Addition with incompatible Elements performed" << endl;
51         exit(1);
52     }
53
54     int t = this->n + x->n;
55
56     return new Integer(t);
57 }
58
59 Ring* Integer::operator*(Ring* c) {
60     Integer* x = dynamic_cast<Integer*>(c);
61
62     //if the cast is unsuccessful, we exit
63     if (x == 0) {
64         cout << "Error: Multiplikation with incompatible Elements performed" << endl;
65         exit(2);
66     }
67
68     return new Integer(this->n * x->n);
69 }
70
71 // comparison function
72 bool Integer::operator==(Ring* c) {
73     Integer* x = dynamic_cast<Integer*>(c);
74
75
76     //if cast is unsuccessful, we exit
77     if (x == 0) {
78         cout << "Error: Comparison with incompatible Elements performed" << endl;
79         exit(3);
80     }
81
82     if (this->n == x->n) {
83         return true;
84     }
85     else {
86         return false;
87     }
88
89 }
```

2.3 The Program - RecPoly

```
1 //*****************************************************************
2 // "RecPoly.h"
3 //
```

```
4  // contains the definition of the class RecPol which is derived from
5  // the class Ring. It implements the Ring of polynomials.
6  //
7  // created by: Felix Dressler - 24.05.2022
8  //*****************************************************************
9
10 #pragma once
11 #include"Ring.h"
12
13 class RecPoly : public Ring {
14 private:
15   Ring** coeff;
16   int n;
17   string var;
18
19 public:
20   // polynomial with n>=0 coefficients and given variable name
21   RecPoly(string var, int n, Ring** coeffs);
22   // copy constructor, copy assignment operator, destructor
23   RecPoly(RecPoly& p);
24   RecPoly& operator=(RecPoly& p);
25   virtual ~RecPoly();
26
27   //virtual functions from Ring:
28
29   // a heap-allocated duplicate of this element
30   virtual Ring* clone();
31
32   // the string representation of this element
33   virtual string str();
34
35   // the constant of the type of this element and the inverse of this element
36   virtual Ring* zero();
37   virtual Ring* operator-();
38
39   // sum and product of this element and c
40   virtual Ring* operator+(Ring* c);
41   virtual Ring* operator*(Ring* c);
42
43   // comparison function
44   virtual bool operator==(Ring* c);
45 };
```

```
1  //*****************************************************************
2  // "RecPoly.cpp"
3  //
4  //
5  //
6  // created by: Felix Dressler - 24.05.2022
7  //*****************************************************************
8
9  #include"RecPoly.h"
10 #include"Integer.h"
11
12 #include<algorithm>
13 #include<iostream>
14
15 using namespace std;
16
```

```
17
18 // polynomial with n>=0 coefficients and given variable name
19 RecPoly::RecPoly(string var, int n, Ring** coeffs) {
20     this->var = var;
21
22     int zeros = 0;
23
24     //cuts of all 0s at the end of the coeffs array
25     if (n != 0) {
26         Ring* z = coeffs[0]->zero();
27
28         for (int i = n - 1; i >= 0; i--) {
29             if (!(coeffs[i]->operator==(z))) {
30                 break;
31             }
32             zeros++;
33         }
34         delete z;
35     }
36
37     this->n = n-zeros;
38     this->coeff = new Ring*[n];
39     for (int i = 0; i < n; i++) {
40         coeff[i] = coeffs[i]->clone();//clone to make sure only we have control over the
               array
41     }
42 }
43
44 // copy constructor, copy assignment operator, destructor
45 RecPoly::RecPoly(RecPoly& p) {
46     this->var = p.var;
47     this->n = p.n;
48
49     this->coeff = new Ring * [n];
50     for (int i = 0; i < n; i++) {
51         coeff[i] = p.coeff[i]->clone();
52     }
53 }
54
55 RecPoly& RecPoly::operator=(RecPoly& p) {
56     this->var = p.var;
57     this->n = p.n;
58
59     delete[] this->coeff;
60     this->coeff = new Ring*[n];
61
62     for (int i = 0; i < n; i++) {
63         coeff[i] = p.coeff[i]->clone();
64     }
65
66     return *this;
67 }
68
69 RecPoly::~RecPoly() {
70     for (int i = 0; i < this->n;i++) {
71         delete coeff[i];
72     }
73     delete[] coeff;
74 }
75
```

```
76  // a heap-allocated duplicate of this element
77  Ring* RecPoly::clone() {
78
79      return new RecPoly(*this);
80  }
81
82  // the string representation of this element
83  string RecPoly::str() {
84      string str = "";
85      if (n == 0) {
86          str = "0";
87      }
88      else {
89          str += "(";
90          for (int i = 0; i < n; i++) {
91              if (!(coeff[i]->operator==(coeff[i]->zero()))) {
92                  str += coeff[i]->str();
93                  if (i != 0) {
94                      str += "*" + var + "^" + to_string(i);
95                  }
96                  if (i < n - 1) {
97                      str += "+";
98                  }
99              }
100         }
101         str += ")";
102     }
103
104     return str;
105 }
106
107 // the constant of the type of this element and the inverse of this element
108 Ring* RecPoly::zero() {
109     return new RecPoly(this->var, 0, {});
110 }
111
112 Ring* RecPoly::operator-() {
113
114     Ring** temp = new Ring*[this->n];
115
116     for (int i = 0; i < this->n; i++) {
117         temp[i] = this->coeff[i]->operator-();
118     }
119
120     RecPoly* ret = new RecPoly(this->var, this->n, temp);
121
122     for (int i = 0; i < this->n; i++) {
123         delete temp[i];
124     }
125     delete[] temp;
126
127     return ret;
128 }
129
130 // sum and product of this element and c
131 Ring* RecPoly::operator+(Ring* c) {
132
133     RecPoly* x = dynamic_cast<RecPoly*>(c);
134
135     if (x == 0) {
```

```
136         cout << "Error: Addition with incompatible Elements performed" << endl;
137         exit(3);
138     }
139     if (this->var != x->var) {
140         cout << "Error: Addition with incompatible Polynomials performed (wrong variables
                )" << endl;
141         exit(4);
142     }
143 
144     else {
145         int n_temp = max(this->n,x->n);
146 
147         Ring** temp = new Ring*[n_temp];
148 
149         if (this->n == 0) {
150             for (int i = 0; i < x->n; i++) {
151                 temp[i] = x->coeff[i]->clone();
152             }
153 
154             RecPoly* add = new RecPoly(this->var, x->n, temp);
155 
156             for (int i = 0; i < n_temp; i++) {
157                 delete temp[i];
158             }
159             delete[] temp;
160 
161             return add;
162         }
163         else {
164 
165             for (int i = 0; i < this->n && i < x->n; i++) {
166                 temp[i] = this->coeff[i]->operator+(x->coeff[i]);
167             }
168 
169             if (this->n > x->n) {
170                 for (int i = x->n; i < this->n; i++) {
171                     temp[i] = this->coeff[i];
172                 }
173             }
174             else if(this->n < x->n) {
175                 for (int i = this->n; i < x->n; i++) {
176                     temp[i] = x->coeff[i];
177                 }
178             }
179 
180             RecPoly* add = new RecPoly(this->var, n_temp, temp);
181 
182             return add;
183         }
184     }
185 
186 
187 }
188 
189 Ring* RecPoly::operator*(Ring* c) {
190 
191     RecPoly* x = dynamic_cast<RecPoly*>(c);
192 
193     if (x == 0) {
194         cout << "Error: Multiplication with incompatible Elements performed" << endl;
```

```
195         exit(5);
196     }
197     if (this->var != x->var) {
198         cout << "Error: Multiplication with incompatible Polynomials performed (wrong
                variables)" << endl;
199         exit(6);
200     }
201 
202     else {
203         if (this->n == 0 || x->n == 0) {
204             return this->zero(); // new RecPoly(this->var, 0, {});
205         }
206         else{
207 
208             int length = this->n + x->n -1;
209 
210             Ring** temp = new Ring*[length];
211 
212             for (int i = 0; i < length; i++) {
213                 temp[i] = x->coeff[0]->zero();
214             }
215 
216             for (int i = 0; i < this->n; i++) {
217                 for (int j = 0; j < x->n; j++) {
218                     Ring* del = temp[i+j];
219                     temp[i+j] = temp[i+j]->operator+(this->coeff[i]->operator*(x->coeff[j
                            ]));
220                     delete del;
221                 }
222 
223             }
224 
225             RecPoly* mult = new  RecPoly(this->var, length, temp);
226 
227             for (int i = 0; i < length; i++) {
228                 delete temp[i];
229             }
230             delete[] temp;
231 
232             return mult;
233         }
234     }
235 
236 }
237 
238 // comparison function
239 bool RecPoly::operator==(Ring* c) {
240 
241     RecPoly* x = dynamic_cast<RecPoly*>(c);
242 
243     bool same = true;
244 
245     for (int i = 0; i < this->n; i++) {
246         if (this->coeff[i] != x->coeff[i]) {
247             same = false;
248         }
249     }
250     return same;
251 }
```