

Programming 2 - Assignment 4

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

May 24, 2022

1 The Program

1.1 The Program - Ring

```
1 #pragma once
2 #include<string>
3
4 using namespace std;
5
6 class Ring {
7 public:
8     // destructor
9     virtual ~Ring() {}
10
11     // a heap-allocated duplicate of this element
12     virtual Ring* clone() = 0;
13
14     // the string representation of this element
15     virtual string str() = 0;
16
17     // the constant of the type of this element and the inverse of this element
18     virtual Ring* zero() = 0;
19     virtual Ring* operator-() = 0;
20
21     // sum and product of this element and c
22     virtual Ring* operator+(Ring* c) = 0;
23     virtual Ring* operator*(Ring* c) = 0;
24
25     // comparison function
26     virtual bool operator==(Ring* c) = 0;
27 };
```

1.2 The Program - Integer

```
1 #pragma once
2 #include"Ring.h"
3
4 class Integer : public Ring {
5 private:
6     int n;
7 public:
8     // integer with value n (default 0)
9     Integer(int n = 0);
10
11     // destructor
12     virtual ~Integer() {
13
14     }
15
16     // a heap-allocated duplicate of this element
17     virtual Ring* clone();
18
19     // the string representation of this element
20     virtual string str();
21
22     // the constant of the type of this element and the inverse of this element
```

```

23  virtual Ring* zero();
24  virtual Ring* operator-();
25
26  // sum and product of this element and c
27  virtual Ring* operator+(Ring* c);
28  virtual Ring* operator*(Ring* c);
29
30  // comparison function
31  virtual bool operator==(Ring* c);
32  };

```

```

1  #include "Integer.h"
2  #include <iostream>
3
4  using namespace std;
5
6  // integer with value n (default 0)
7  Integer::Integer(int n) {
8      this->n = n;
9  }
10
11 // a heap-allocated duplicate of this element
12 Ring* Integer::clone() {
13     Integer* c = new Integer(this->n);
14
15     return c;
16 }
17
18 // the string representation of this element
19 string Integer::str() {
20
21     return to_string(this->n);
22 }
23
24 // the constant of the type of this element and the inverse of this element
25 Ring* Integer::zero() {
26
27     return new Integer(0);
28 }
29
30 Ring* Integer::operator-() {
31
32     return new Integer(-(this->n));
33 }
34
35 // sum and product of this element and c
36 Ring* Integer::operator+(Ring* c) {
37     Integer* x = dynamic_cast<Integer*>(c);
38
39     if (x == 0) {
40         cout << "Error: Addition with incompatible Elements performed" << endl;
41         exit(1);
42     }
43
44     this->n += x->n;
45
46     return this; // like this
47 }
48

```

```

49 Ring* Integer::operator*(Ring* c) {
50     Integer* x = dynamic_cast<Integer*>(c);
51
52     if (x == 0) {
53         cout << "Error: Multiplikation with incompatible Elements performed" << endl;
54         exit(2);
55     }
56
57     return new Integer(this->n * x->n);
58 }
59
60 // comparison function
61 bool Integer::operator==(Ring* c) {
62     Integer* x = dynamic_cast<Integer*>(c);
63
64     if (x == 0) {
65         cout << "Error: Comparison with incompatible Elements performed" << endl;
66         exit(3);
67     }
68
69     if (this->n == x->n) {
70         return true;
71     }
72     else {
73         return false;
74     }
75 }
76 }

```

1.3 The Program - RecPoly

```

1  #pragma once
2  #include "Ring.h"
3
4  class RecPoly : public Ring {
5  private:
6      Ring** coeff;
7      int n;
8      string var;
9
10 public:
11     // polynomial with n>=0 coefficients and given variable name
12     RecPoly(string var, int n, Ring** coeffs);
13     // copy constructor, copy assignment operator, destructor
14     RecPoly(RecPoly& p);
15     RecPoly& operator=(RecPoly& p);
16     virtual ~RecPoly();
17
18     // a heap-allocated duplicate of this element
19     virtual Ring* clone();
20
21     // the string representation of this element
22     virtual string str();
23
24     // the constant of the type of this element and the inverse of this element
25     virtual Ring* zero();
26     virtual Ring* operator-();
27

```

```

28 // sum and product of this element and c
29 virtual Ring* operator+(Ring* c);
30 virtual Ring* operator*(Ring* c);
31
32 // comparison function
33 virtual bool operator==(Ring* c);
34 };

```

```

1 #include "RecPoly.h"
2 #include "Integer.h"
3
4 #include <iostream>
5
6 using namespace std;
7
8
9 // polynomial with n>=0 coefficients and given variable name
10 RecPoly::RecPoly(string var, int n, Ring** coeffs) {
11     this->var = var;
12     this->n = n;
13     this->coeff = new Ring*[n];
14     for (int i = 0; i < n; i++) {
15         coeff[i] = coeffs[i]->clone(); //clone to make sure only we have control over the
            array
16     }
17 }
18
19 // copy constructor, copy assignment operator, destructor
20 RecPoly::RecPoly(RecPoly& p) {
21     this->var = p.var;
22     this->n = p.n;
23
24     this->coeff = new Ring * [n];
25     for (int i = 0; i < n; i++) {
26         coeff[i] = p.coeff[i]->clone();
27     }
28 }
29
30 RecPoly& RecPoly::operator=(RecPoly& p) {
31     this->var = p.var;
32     this->n = p.n;
33
34     this->coeff = new Ring * [n];
35     for (int i = 0; i < n; i++) {
36         coeff[i] = p.coeff[i]->clone();
37     }
38
39     return *this;
40 }
41
42 RecPoly::~RecPoly() {
43     for (int i = 0; i < this->n; i++) {
44         delete coeff[i];
45     }
46     delete[] coeff;
47 }
48
49 // a heap-allocated duplicate of this element
50 Ring* RecPoly::clone() {

```

```

51     return new RecPoly(*this);
52 }
53
54 // the string representation of this element
55 string RecPoly::str() {
56     string str = "";
57     if (n == 0) {
58         str = "0";
59     }
60     else {
61         str += "(";
62         for (int i = 0; i < n; i++) {
63             if (!(coeff[i]->operator==(coeff[i]->zero()))) {
64                 str += coeff[i]->str() + "*" + var + "^" + to_string(i);
65                 if (i < n - 1) {
66                     str += "+";
67                 }
68             }
69         }
70         str += ")";
71     }
72     return str;
73 }
74
75 // the constant of the type of this element and the inverse of this element
76 Ring* RecPoly::zero() {
77     return new RecPoly(this->var, 0, {});
78 }
79
80 Ring* RecPoly::operator-() {
81     for (int i = 0; i < this->n; i++) {
82         this->coeff[i] = this->coeff[i]->operator-();
83     }
84     return this;
85 }
86
87 // sum and product of this element and c
88 Ring* RecPoly::operator+(Ring* c) {
89     RecPoly* x = dynamic_cast<RecPoly*>(c);
90
91     if (x == 0) {
92         cout << "Error: Addition with incompatible Elements performed" << endl;
93         exit(3);
94     }
95     else {
96         int n_temp = 0;
97
98         if (this->n >= x->n) {
99             n_temp = this->n;
100         }
101         else {
102             n_temp = x->n;
103         }
104
105         Ring** temp = new Ring * [n_temp];

```

```

111
112     if (this->n == 0) {
113         for (int i = 0; i < x->n; i++) {
114             temp[i] = x->coeff[i]->clone();
115         }
116
117         RecPoly* add = new RecPoly(this->var, x->n, temp);
118
119         //delete
120
121         return add;
122     }
123     else {
124
125         for (int i = 0; i < this->n && i < x->n; i++) {
126             temp[i] = this->coeff[i]->operator+(x->coeff[i]);
127         }
128
129
130         if (this->n > x->n) {
131             for (int i = x->n; i < this->n; i++) {
132                 temp[i] = this->coeff[i]->operator+(x->coeff[i]); //why???
133             }
134         }
135         else if (this->n < x->n) {
136             for (int i = this->n; i < x->n; i++) {
137                 temp[i] = this->coeff[i]->operator+(x->coeff[i]);
138             }
139         }
140         //for (int i = 0; i < this->n; i++) {
141         //    temp[i] = c->operator+(this->coeff[i]); //important
142         //}
143
144         RecPoly* add = new RecPoly(this->var, this->n, temp);
145
146         for (int i = 0; i < n_temp; i++) {
147             delete temp[i];
148         }
149         delete[] temp;
150
151         return add;
152     }
153 }
154
155
156 }
157
158 Ring* RecPoly::operator*(Ring* c) {
159
160     RecPoly* x = dynamic_cast<RecPoly*>(c);
161
162     if (x == 0) {
163         cout << "Error: Multiplication with incompatible Elements performed" << endl;
164         exit(4);
165     }
166     if (this->var != x->var) {
167         cout << "Error: Multiplication with incompatible Polynomials performed (wrong
168             variables)" << endl;
169         exit(4);
170     }

```

```

170
171     else {
172         if (this->n == 0 || x->n == 0) {
173             return new RecPoly(this->var, 0, {});
174         }
175         else{
176             Ring** temp = new Ring*[this->n + x->n];
177
178             for (int i = 0; i < this->n + x->n; i++) {
179                 //temp[i] = 0;
180                 for (int j = 0; j < i; j++) {
181                     temp[i] = *temp[i] + *this->coeff[i] * x->coeff[i];
182                 } //->operator ->operator
183             }
184
185             RecPoly* mult = new RecPoly(this->var, this->n, temp);
186
187             //for (int i = 0; i < this->n; i++) {
188             //    delete temp[i];
189             //}
190             //delete[] temp;
191
192             return mult;
193         }
194     }
195
196 }
197
198 // comparison function
199 bool RecPoly::operator==(Ring* c) {
200
201     for (int i = 0; i < this->n; i++) {
202         if (this->coeff[i] != c) {
203             return false;
204         }
205     }
206     return true;
207 }

```

1.4 The Program - Main

```

1  #include "Integer.h"
2  #include "RecPoly.h"
3  #include <iostream>
4
5  using namespace std;
6
7  int main() {
8      Ring* c[] = { new Integer(-5), new Integer(2), new Integer(0), new Integer(-3) };
9
10     RecPoly* p = new RecPoly("x", 4, c); // p = -3x^3 + 2x - 5
11
12     cout << p->str() << endl;
13
14     RecPoly* q = //new RecPoly(*p); // = // q = p+p = -6x^3 + 4x - 10
15         dynamic_cast<RecPoly*>(p->operator+(p));
16     cout << q->str() << endl;
17     RecPoly* r = // r = p*q

```



```
18     dynamic_cast<RecPoly*>(p->operator*(q));  
19     cout << r->str() << endl;  
20  
21  
22  
23     return 0;  
24 }
```