Programming 2 - Assignment 5

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

June 3, 2022

1 Testing the Program

1.1 Main.cpp

For testing the *Main.cpp* was adapted to work with the new templates.

```
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     Integer* c[] = { new Integer(-5), new Integer(2), new Integer(0), new Integer(-3) };
18
19     UniPoly* p = new UniPoly("x", 4, c); // p = -3x^3 + 2x - 5
20     cout << p->str() << endl;
21
22     UniPoly* q = // q = p+p = -6x^3 + 4x - 10
23         p->operator+(p);
24     cout << q->str() << endl;
25
26     UniPoly* r = // r = p*q = 50 - 40 x + 8 x^2 + 60 x^3 - 24 x^4 + 18 x^6
27         p->operator*(q);
28     cout << r->str() << endl;
29
30     //additional tests
31
32     //zero
33
34     UniPoly* z = //zero polynomial
35         p->zero();
36     cout << z->str() << endl;
37
38     UniPoly* z2 = // = p + 0
39         p->operator+(z);
40     cout << z2->str() << endl;
41
42     UniPoly* z3 = // = p * 0
43         p->operator*(z);
44     cout << z3->str() << endl;
45
46     //negation
47
48     UniPoly* g = //= -p
49         p->operator-();
50     cout << g->str() << endl;
51
52     UniPoly* g2 = // = -p + p = 0
53         g->operator+(p);
```

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

1.2 output

2 RecPoly.h

```
1   #pragma once
2
3   #include<string>
4   #include<iostream>
5
6   #include"Integer.h"
7
8
9   using namespace std;
10
11  template<class Ring> class RecPoly
12  {
13  private:
14      Ring** coeff;
15      int n;
16      string var;
17
18  public:
19
20      // polynomial with n>=0 coefficients and given variable name
21      RecPoly(string var, int n, Ring** coeffs) {
22          this->var = var;
23
24          int zeros = 0;
25
26          //cuts of all 0s at the end of the coeffs array
27          if (n != 0) {
28              Ring* z = coeffs[0]->zero();
29
```

```
30              for (int i = n - 1; i >= 0; i--) {
31                  if (!(coeffs[i]->operator==(z))) {
32                      break;
33                  }
34                  zeros++;
35              }
36              delete z;
37          }
38
39          this->n = n - zeros;
40          this->coeff = new Ring * [n];
41          for (int i = 0; i < n; i++) {
42              coeff[i] = coeffs[i]->clone();//clone to make sure only we have control over
                    the array
43          }
44      }
45
46      // copy constructor, copy assignment operator
47      RecPoly(RecPoly& p) {
48          this->var = p.var;
49          this->n = p.n;
50
51          this->coeff = new Ring * [n];
52          for (int i = 0; i < n; i++) {
53              coeff[i] = p.coeff[i]->clone();
54          }
55      }
56
57      RecPoly& operator=(RecPoly& p) {
58          this->var = p.var;
59          this->n = p.n;
60
61          delete[] this->coeff;
62          this->coeff = new Ring * [n];
63
64          for (int i = 0; i < n; i++) {
65              coeff[i] = p.coeff[i]->clone();
66          }
67
68          return *this;
69      }
70
71      //destructor for RecPoly
72      ~RecPoly() {
73          for (int i = 0; i < this->n; i++) {
74              delete coeff[i];
75          }
76          delete[] coeff;
77      }
78
79      // a heap-allocated duplicate of this element
80      RecPoly* clone() {
81
82          return new RecPoly(*this);
83      }
84
85      // the string representation of this element
86      string str() {
87          string str = "";
88          if (n == 0) {
```

```
89              str = "0";
90          }
91          else {
92              str += "(";
93              for (int i = 0; i < n; i++) {
94                  if (!(coeff[i]->operator==(coeff[i]->zero()))) {
95                      str += coeff[i]->str();
96                      if (i != 0) {
97                          str += "*" + var + "^" + to_string(i);
98                      }
99                      if (i < n - 1) {
100                         str += "+";
101                     }
102                 }
103             }
104             str += ")";
105         }
106
107         return str;
108     }
109
110     // the constant of the type of this element and the inverse of this element
111     RecPoly* zero() {
112         return new RecPoly(this->var, 0, {});
113     }
114
115     RecPoly* operator-() {
116
117         Ring** temp = new Ring * [this->n];
118
119         for (int i = 0; i < this->n; i++) {
120             temp[i] = this->coeff[i]->operator-();
121         }
122
123         RecPoly* ret = new RecPoly(this->var, this->n, temp);
124
125         for (int i = 0; i < this->n; i++) {
126             delete temp[i];
127         }
128         delete[] temp;
129
130         return ret;
131     }
132
133     // sum operator for polynomials
134     RecPoly* operator+(RecPoly* x) {
135         if (this->var != x->var) {
136             cout << "Error: Addition with incompatible Polynomials performed (wrong
                    variables)" << endl;
137             exit(4);
138         }
139
140         else {
141             int n_temp = max(this->n, x->n);
142
143             Ring** temp = new Ring * [n_temp];
144
145             if (this->n == 0) {
146                 for (int i = 0; i < x->n; i++) {
147                     temp[i] = x->coeff[i]->clone();
```

```
148                 }
149
150                 RecPoly* add = new RecPoly(this->var, x->n, temp);
151
152                 for (int i = 0; i < n_temp; i++) {
153                     delete temp[i];
154                 }
155                 delete[] temp;
156
157                 return add;
158             }
159             else {
160
161                 for (int i = 0; i < this->n && i < x->n; i++) {
162                     temp[i] = this->coeff[i]->operator+(x->coeff[i]);
163                 }
164
165                 if (this->n > x->n) {
166                     for (int i = x->n; i < this->n; i++) {
167                         temp[i] = this->coeff[i];
168                     }
169                 }
170                 else if (this->n < x->n) {
171                     for (int i = this->n; i < x->n; i++) {
172                         temp[i] = x->coeff[i];
173                     }
174                 }
175
176                 RecPoly* add = new RecPoly(this->var, n_temp, temp);
177
178                 //for (int i = 0; i < n_temp; i++) {
179                 //    delete temp[i];
180                 //}
181                 //delete[] temp;
182
183                 return add;
184             }
185         }
186     }
187
188
189     // multiplication operator for polynomials
190     RecPoly* operator*(RecPoly* x) {
191         if (this->var != x->var) {
192             cout << "Error: Multiplication with incompatible Polynomials performed (wrong
                      variables)" << endl;
193             exit(6);
194         }
195
196         else {
197             if (this->n == 0 || x->n == 0) {
198                 return this->zero(); // new RecPoly(this->var, 0, {});
199             }
200             else {
201
202                 int length = this->n + x->n - 1;
203
204                 Ring** temp = new Ring * [length];
205
206                 for (int i = 0; i < length; i++) {
```

```
207                     temp[i] = x->coeff[0]->zero();
208                 }
209 
210                 for (int i = 0; i < this->n; i++) {
211                     for (int j = 0; j < x->n; j++) {
212                         Ring* del = temp[i + j];
213                         temp[i + j] = temp[i + j]->operator+(this->coeff[i]->operator*(x
                                ->coeff[j]));
214                         delete del;
215                     }
216 
217                 }
218 
219                 RecPoly* mult = new  RecPoly(this->var, length, temp);
220 
221                 for (int i = 0; i < length; i++) {
222                     delete temp[i];
223                 }
224                 delete[] temp;
225 
226                 return mult;
227             }
228         }
229 
230     }
231 
232     // comparison function for Plynomials
233     bool operator==(RecPoly* x) {
234         bool same = true;
235 
236         for (int i = 0; i < this->n; i++) {
237             if (this->coeff[i] != x->coeff[i]) {
238                 same = false;
239             }
240         }
241         return same;
242     }
243 };
244 
245 typedef RecPoly<Integer> UniPoly;
246 
247 typedef RecPoly<UniPoly> BiPoly;
```

3 Integer

3.1 Integer.h

```
1 #pragma once
2 
3 #include<string>
4 #include<iostream>
5 
6 using namespace std;
7 
8 class Integer {
9 private:
```

```
10   int n;
11 public:
12   // integer with value n (default 0)
13   Integer(int n = 0);
14
15   // destructor - empty because in Integer no new arrays/pointers are created
16   ~Integer() {
17
18   }
19
20   // a heap-allocated duplicate of this element
21   Integer* clone();
22
23   // the string representation of this element
24   string str();
25
26   // the constant of the type of this element and the inverse of this element
27   Integer* zero();
28   Integer* operator-();
29
30   // sum and product of this element and c
31   Integer* operator+(Integer* c);
32   Integer* operator*(Integer* c);
33
34   // comparison function
35   bool operator==(Integer* c);
36 };
```

3.2 Integer.cpp

```
1  #include"Integer.h"
2
3  // integer with value n (default 0)
4  Integer::Integer(int n) {
5      this->n = n;
6  }
7
8  // a heap-allocated duplicate of this element
9  Integer* Integer::clone() {
10     Integer* c = new Integer(this->n);
11
12     return c;
13 }
14
15 // the string representation of this element
16 string Integer::str() {
17
18     return to_string(this->n);;
19 }
20
21 // the constant of the type of this element and the inverse of this element
22 Integer* Integer::zero() {
23
24     return new Integer(0);
25 }
26
27 Integer* Integer::operator-() {
28
```

```
29      return new Integer(-(this->n));
30  }
31
32  // sum and product of this element and c
33  Integer* Integer::operator+(Integer* x) {
34
35      int t = this->n + x->n;
36
37      return new Integer(t);
38  }
39
40  Integer* Integer::operator*(Integer* x) {
41
42      return new Integer(this->n * x->n);
43  }
44
45  // comparison function
46  bool Integer::operator==(Integer* x) {
47
48      if (this->n == x->n) {
49          return true;
50      }
51      else {
52          return false;
53      }
54
55  }
```