Programming 2 - Assignment 5

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

June 7, 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 the template class RecPoly.
5  //
6  // created by: Felix Dressler - 03.06.2022
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

The output was the same as in the previous project.

```
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 RecPoly.h

Because this project was about using the previous code and adapt it to work with templates, the memory leak in line 186-189 still exists.

```
1  //****************************************************************
2  // "RecPoly.h"
3  //
4  // contains the template class RecPoly and its functions.
5  //
6  // created by: Felix Dressler - 03.06.2022
7  //****************************************************************
8
9  #pragma once
10
11 #include<string>
12 #include<iostream>
13
14 #include"Integer.h"
15
16
17 using namespace std;
18
19 template<class Ring> class RecPoly
20 {
21 private:
22     Ring** coeff;
23     int n;
24     string var;
25
26 public:
27
28     // polynomial with n>=0 coefficients and given variable name
29     RecPoly(string var, int n, Ring** coeffs) {
30         this->var = var;
31
32         int zeros = 0;
33
34         //cuts of all 0s at the end of the coeffs array
35         if (n != 0) {
36             Ring* z = coeffs[0]->zero();
37
38             for (int i = n - 1; i >= 0; i--) {
39                 if (!(coeffs[i]->operator==(z))) {
40                     break;
41                 }
42                 zeros++;
43             }
44             delete z;
45         }
46
47         this->n = n - zeros;
48         this->coeff = new Ring * [n];
49         for (int i = 0; i < n; i++) {
50             coeff[i] = coeffs[i]->clone();//clone to make sure only we have control over
                the array
51         }
52     }
53
```

```
54      // copy constructor, copy assignment operator
55      RecPoly(RecPoly& p) {
56          this->var = p.var;
57          this->n = p.n;
58
59          this->coeff = new Ring * [n];
60          for (int i = 0; i < n; i++) {
61              coeff[i] = p.coeff[i]->clone();
62          }
63      }
64
65      RecPoly& operator=(RecPoly& p) {
66          this->var = p.var;
67          this->n = p.n;
68
69          delete[] this->coeff;
70          this->coeff = new Ring * [n];
71
72          for (int i = 0; i < n; i++) {
73              coeff[i] = p.coeff[i]->clone();
74          }
75
76          return *this;
77      }
78
79      //destructor for RecPoly
80      ~RecPoly() {
81          for (int i = 0; i < this->n; i++) {
82              delete coeff[i];
83          }
84          delete[] coeff;
85      }
86
87      // a heap-allocated duplicate of this element
88      RecPoly* clone() {
89
90          return new RecPoly(*this);
91      }
92
93      // the string representation of this element
94      string str() {
95          string str = "";
96          if (n == 0) {
97              str = "0";
98          }
99          else {
100             str += "(";
101             for (int i = 0; i < n; i++) {
102                 if (!(coeff[i]->operator==(coeff[i]->zero()))) {
103                     str += coeff[i]->str();
104                     if (i != 0) {
105                         str += "*" + var + "^" + to_string(i);
106                     }
107                     if (i < n - 1) {
108                         str += "+";
109                     }
110                 }
111             }
112             str += ")";
113         }
```

```
114
115         return str;
116     }
117
118     // the constant of the type of this element and the inverse of this element
119     RecPoly* zero() {
120         return new RecPoly(this->var, 0, {});
121     }
122
123     RecPoly* operator-() {
124
125         Ring** temp = new Ring * [this->n];
126
127         for (int i = 0; i < this->n; i++) {
128             temp[i] = this->coeff[i]->operator-();
129         }
130
131         RecPoly* ret = new RecPoly(this->var, this->n, temp);
132
133         for (int i = 0; i < this->n; i++) {
134             delete temp[i];
135         }
136         delete[] temp;
137
138         return ret;
139     }
140
141     // sum operator for polynomials
142     RecPoly* operator+(RecPoly* x) {
143         if (this->var != x->var) {
144             cout << "Error: Addition with incompatible Polynomials performed (wrong
                    variables)" << endl;
145             exit(4);
146         }
147
148         else {
149             int n_temp = max(this->n, x->n);
150
151             Ring** temp = new Ring * [n_temp];
152
153             if (this->n == 0) {
154                 for (int i = 0; i < x->n; i++) {
155                     temp[i] = x->coeff[i]->clone();
156                 }
157
158                 RecPoly* add = new RecPoly(this->var, x->n, temp);
159
160                 for (int i = 0; i < n_temp; i++) {
161                     delete temp[i];
162                 }
163                 delete[] temp;
164
165                 return add;
166             }
167             else {
168
169                 for (int i = 0; i < this->n && i < x->n; i++) {
170                     temp[i] = this->coeff[i]->operator+(x->coeff[i]);
171                 }
172
```

```
173                 if (this->n > x->n) {
174                     for (int i = x->n; i < this->n; i++) {
175                         temp[i] = this->coeff[i];
176                     }
177                 }
178                 else if (this->n < x->n) {
179                     for (int i = this->n; i < x->n; i++) {
180                         temp[i] = x->coeff[i];
181                     }
182                 }
183 
184                 RecPoly* add = new RecPoly(this->var, n_temp, temp);
185 
186                 //for (int i = 0; i < n_temp; i++) {
187                 //    delete temp[i];
188                 //}
189                 //delete[] temp;
190 
191                 return add;
192             }
193         }
194     }
195 
196 
197     // multiplication operator for polynomials
198     RecPoly* operator*(RecPoly* x) {
199         if (this->var != x->var) {
200             cout << "Error: Multiplication with incompatible Polynomials performed (wrong
                      variables)" << endl;
201             exit(6);
202         }
203 
204         else {
205             if (this->n == 0 || x->n == 0) {
206                 return this->zero(); // new RecPoly(this->var, 0, {});
207             }
208             else {
209 
210                 int length = this->n + x->n - 1;
211 
212                 Ring** temp = new Ring * [length];
213 
214                 for (int i = 0; i < length; i++) {
215                     temp[i] = x->coeff[0]->zero();
216                 }
217 
218                 for (int i = 0; i < this->n; i++) {
219                     for (int j = 0; j < x->n; j++) {
220                         Ring* del = temp[i + j];
221                         temp[i + j] = temp[i + j]->operator+(this->coeff[i]->operator*(x
                                ->coeff[j]));
222                         delete del;
223                     }
224 
225                 }
226 
227                 RecPoly* mult = new  RecPoly(this->var, length, temp);
228 
229                 for (int i = 0; i < length; i++) {
230                     delete temp[i];
```

```
231                 }
232                 delete[] temp;
233 
234                 return mult;
235             }
236         }
237 
238     }
239 
240     // comparison function for Plynomials
241     bool operator==(RecPoly* x) {
242         bool same = true;
243 
244         for (int i = 0; i < this->n; i++) {
245             if (this->coeff[i] != x->coeff[i]) {
246                 same = false;
247             }
248         }
249         return same;
250     }
251 };
252 
253 typedef RecPoly<Integer> UniPoly;
254 
255 typedef RecPoly<UniPoly> BiPoly;
```

3 Integer

3.1 Integer.h

```
1  //*****************************************************************
2  // "Integer.h"
3  //
4  // contains the class Integer and its function deklarations.
5  //
6  // created by: Felix Dressler - 03.06.2022
7  //*****************************************************************
8
9  #pragma once
10
11 #include<string>
12 #include<iostream>
13
14 using namespace std;
15
16 class Integer {
17 private:
18   int n;
19 public:
20   // integer with value n (default 0)
21   Integer(int n = 0);
22
23   // destructor - empty because in Integer no new arrays/pointers are created
24   ~Integer() {
25
26   }
27
28   // a heap-allocated duplicate of this element
29   Integer* clone();
30
31   // the string representation of this element
32   string str();
33
34   // the constant of the type of this element and the inverse of this element
35   Integer* zero();
36   Integer* operator-();
37
38   // sum and product of this element and c
39   Integer* operator+(Integer* c);
40   Integer* operator*(Integer* c);
41
42   // comparison function
43   bool operator==(Integer* c);
44 };
```

3.2 Integer.cpp

```
1  //*****************************************************************
2  // "Integer.cpp"
3  //
4  // contains the function definitions of the class Integer.
5  //
```

```
6  // created by: Felix Dressler - 03.06.2022
7  //*****************************************************************
8
9  #include"Integer.h"
10
11 // integer with value n (default 0)
12 Integer::Integer(int n) {
13     this->n = n;
14 }
15
16 // a heap-allocated duplicate of this element
17 Integer* Integer::clone() {
18     Integer* c = new Integer(this->n);
19
20     return c;
21 }
22
23 // the string representation of this element
24 string Integer::str() {
25
26     return to_string(this->n);;
27 }
28
29 // the constant of the type of this element and the inverse of this element
30 Integer* Integer::zero() {
31
32     return new Integer(0);
33 }
34
35 Integer* Integer::operator-() {
36
37     return new Integer(-(this->n));
38 }
39
40 // sum and product of this element and c
41 Integer* Integer::operator+(Integer* x) {
42
43     int t = this->n + x->n;
44
45     return new Integer(t);
46 }
47
48 Integer* Integer::operator*(Integer* x) {
49
50     return new Integer(this->n * x->n);
51 }
52
53 // comparison function
54 bool Integer::operator==(Integer* x) {
55
56     if (this->n == x->n) {
57         return true;
58     }
59     else {
60         return false;
61     }
62
63 }
```