# Programming 2 - Assignment 5

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

June 9, 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

This code is almost identical to the code from the last project, except for the change from heritage (and casts) to templates. Also the correction of the previous code was implemented.

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

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

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

```
231
232                 RecPoly* mult = new  RecPoly(this->var, length, temp);
233
234                 for (int i = 0; i < length; i++) {
235                     delete temp[i];
236                 }
237                 delete[] temp;
238
239                 return mult;
240             }
241         }
242
243     }
244
245     // comparison function for Plynomials
246     bool operator==(RecPoly* x) {
247         bool same = true;
248
249         for (int i = 0; i < this->n; i++) {
250             if (this->coeff[i] != x->coeff[i]) {
251                 same = false;
252             }
253         }
254         return same;
255     }
256 };
257
258 typedef RecPoly<Integer> UniPoly;
259
260 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 }
```