Programming 2 - Assignment 2

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

April 28, 2022

1 Testing the Program

For testing the Program, or in specific, the class, a series of tests was performed by testing different methods of this through different main-methods.

1.1 testing the specified commands

In this section the commands given in the assignment instructions will be tested.

The following code-block shows the methods used to perform the first test. As shown, every operation was performed in two variables, with multiple inclusions of both *add()* methods and the *println()* method.

```
1  //*****************************************************************
2  // "Main.cpp"
3  //
4  // is the Main cpp file that was used to test different scenarios
5  // of using the DistPoly class.
6  //
7  // There are different versions of this file with different intentions
8  // in the assignment folder.
9  //
10 //created by Felix Dressler, 28.04.2022
11 //*****************************************************************
12 #include"DistPoly.h"
13 #include<string>
14
15 string vars[2] = { "x","y" };
16
17 int main() {
18
19   // some exponent vectors("power products")
20   int e1[2] = { 1,2 }; int e2[2] = { 2,1 }; int e3[2] = { 1,0 };
21   int e4[2] = { 0,1 }; int e5[2] = { 0,0 }; int e6[2] = { 2,2 };
22
23   // construct zero polynomial in two variables, then add monomials
24   string vars[2] = { "x","y" };
25   DistPoly p(2, vars);
26   p.add(3, e1).add(5, e2).add(7, e3).add(11, e4).add(13, e5);
27
28   // construct zero polynomial in two variables, then add monomials
29   DistPoly q(2, vars);
30   q.add(11, e4).add(-3, e2).add(2, e6).add(-2, e2);
31
32   // print p and q
33   p.println();
34   q.println();
35
36   // set p to p+2*q and print it
37
38   DistPoly r = p;
39   r.add(q).add(q);
40   p = r;
41   p.println();
42
43   return 0;
44 }
```

This is the output, that was created by the code above.

```
5x^2y+3xy^2+7x+11y+13
2x^2y^2-5x^2y+11y
4x^2y^2-5x^2y+3xy^2+7x+33y+13
```

1.2 testing in three and one variable

In this section, tests of the class in one and three variables will be presented. In order to produce results that are comparable we modified the test case from the previous section to work with uni- and three-variate polynomials. By modifying it further, adding zero-polynomials was also tested.

1.2.1 testing in one variable

The following code was used to perform the tests.

```
1  //*************************************************************
2  // "Main.cpp"
3  //
4  // is the Main cpp file that was used to test different scenarios
5  // of using the DistPoly class.
6  //
7  // There are different versions of this file with different intentions
8  // in the assignment folder.
9  //
10 //created by Felix Dressler, 28.04.2022
11 //*************************************************************
12 #include"DistPoly.h"
13 #include<string>
14
15 string vars[1] = { "x" };
16
17 int main() {
18
19   // some exponent vectors("power products")
20   int e1[1] = { 1 }; int e2[1] = { 2 }; int e3[1] = { 1 };
21   int e4[1] = { 0 }; int e5[1] = { 0 }; int e6[1] = { 2 };
22
23   // construct zero polynomial in two variables, then add monomials
24   DistPoly p(1, vars);
25   p.add(3, e1).add(5, e2).add(7, e3).add(11, e4).add(13, e5);
26
27   // construct zero polynomial in two variables, then add monomials
28   DistPoly q(1, vars);
29   q.add(11, e4).add(-3, e2).add(2, e6).add(-2, e2);
30
31   // print p and q
32   p.println();
33   q.println();
34
35   // set p to p+2*q and print it
36
37   DistPoly r = p;
38   r.add(q).add(q);
39   p = r;
40   p.println();
41
42   return 0;
43 }
```

This is the output, that was created by the code above.

```
5x^2+10x+24
-3x^2+11
-1x^2+10x+46
```

1.2.2 testing in three variables

The following code was used to perform the tests.

```
1  //*****************************************************************
2  // "Main.cpp"
3  //
4  // is the Main cpp file that was used to test different scenarios
5  // of using the DistPoly class.
6  //
7  // There are different versions of this file with different intentions
8  // in the assignment folder.
9  //
10 //created by Felix Dressler, 28.04.2022
11 //*****************************************************************
12 #include"DistPoly.h"
13 #include<string>
14
15 string vars3[3] = { "x","y","z" };
16
17 int main() {
18
19   // some exponent vectors("power products")
20   int e1[3] = { 1,2,2 }; int e2[3] = { 2,1,0 }; int e3[3] = { 1,0,0 };
21   int e4[3] = { 0,1,3 }; int e5[3] = { 0,0,0 }; int e6[3] = { 2,2,1 };
22
23   // construct zero polynomial in two variables, then add monomials
24   DistPoly p(3, vars3);
25   p.add(3, e1).add(5, e2).add(7, e3).add(11, e4).add(13, e5);
26
27   // construct zero polynomial in two variables, then add monomials
28   DistPoly q(3, vars3);
29   q.add(11, e4).add(-3, e2).add(2, e6).add(-2, e2);
30
31   // print p and q
32   p.println();
33   q.println();
34
35   // set p to p+2*q and print it
36
37   DistPoly r = p;
38   r.add(q).add(q);
39   p = r;
40   p.println();
41
42   return 0;
43 }
```

This is the output, that was created by the code above.

```
5x^2y+3xy^2z^2+7x+11yz^3+13
2x^2y^2z-5x^2y+11yz^3
4x^2y^2z-5x^2y+3xy^2z^2+7x+33yz^3+13
```

1.2.3 adding the zero-polynomial

The following code was used to perform the tests.

```
//****************************************************************
// "Main.cpp"
//
// is the Main cpp file that was used to test different scenarios
// of using the DistPoly class.
//
// There are different versions of this file with different intentions
// in the assignment folder.
//
//created by Felix Dressler, 28.04.2022
//****************************************************************
#include"DistPoly.h"
#include<string>

string vars3[3] = { "x","y","z" };

int main() {

  // some exponent vectors("power products")
  int e1[3] = { 1,2,2 }; int e2[3] = { 2,1,0 }; int e3[3] = { 1,0,0 };
  int e4[3] = { 0,1,3 }; int e5[3] = { 0,0,0 }; int e6[3] = { 2,2,1 };

  // construct zero polynomial in two variables, then add monomials
  DistPoly p(3, vars3);
  p.println();
  p.add(3, e1).add(5, e2).add(7, e3).add(11, e4).add(13, e5);

  // construct zero polynomial in two variables, then add monomials
  DistPoly q(3, vars3);

  // print p and q
  p.println();
  q.println();

  // set p to p+2*q and print it

  p.add(q);
  p.println();

  return 0;
}
```

This is the output, that was created by the code above.

```
0
5x^2y+3xy^2z^2+7x+11yz^3+13
0
5x^2y+3xy^2z^2+7x+11yz^3+13
```

1.3 testing error messages

In this section, we will test different kinds of errors that can occur during programming with this class. We will try to produce error messages.

1.3.1 adding polynomials with different numbers of variables

The following code was used to perform the tests.

```
1  //*************************************************************
2  // "Main.cpp"
3  //
4  // is the Main cpp file that was used to test different scenarios
5  // of using the DistPoly class.
6  //
7  // There are different versions of this file with different intentions
8  // in the assignment folder.
9  //
10 //created by Felix Dressler, 28.04.2022
11 //*************************************************************
12 #include"DistPoly.h"
13 #include<string>
14
15 string vars2[2] = { "x","y" };
16 string vars3[3] = { "x","y","z" };
17
18 int main() {
19
20   // some exponent vectors("power products")
21   int e1[3] = { 1,2,3 };
22   int e2[2] = { 2,1 };
23
24
25   // construct zero polynomial in three variables, then add monomials
26   DistPoly p(3, vars3);
27   p.add(3, e1);
28
29   DistPoly q(2, vars2);
30   q.add(5, e2);
31
32   //add to polynomials whose vars do not match
33   p.add(q);
34
35   p.println();
36
37   return 0;
38 }
```

This is the output, that was created by the code above. The desired error message has been printed successfully.

```
Error: the number of variables of two added polynomials is differentError: the variables
    of two added polynomials do not match
```

1.3.2 adding polynomials with different orders of variables

The following code was used to perform the tests.

```cpp
//*************************************************************
// "Main.cpp"
//
// is the Main cpp file that was used to test different scenarios
// of using the DistPoly class.
//
// There are different versions of this file with different intentions
// in the assignment folder.
//
//created by Felix Dressler, 28.04.2022
//*************************************************************
#include"DistPoly.h"
#include<string>

string vars2[2] = { "x","y" };
string vars3[3] = { "y","x"};

int main() {

  // some exponent vectors("power products")
  int e1[2] = { 1,2 };
  int e2[2] = { 2,1 };


  // construct zero polynomial in three variables, then add monomials
  DistPoly p(2, vars3);
  p.add(3, e1);

  DistPoly q(2, vars2);
  q.add(5, e2);

  //add to polynomials whose vars do not match
  p.add(q);

  p.println();

  return 0;
}
```

This is the output, that was created by the code above. The desired error message has been printed successfully.

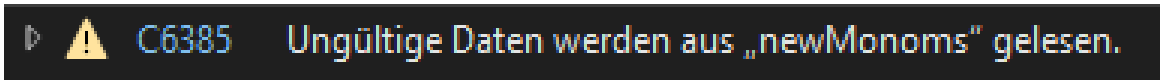
```
Error: the variables of two added polynomials do not match
```

2 Problems

This section will briefly discuss the Problems that have occurred during programming.

2.1 warnings

In the *resize* method, line 283 of the *DistPoly.cpp* this warning is displayed:




```
281 void DistPoly::resize(int factor) {
282     if (factor > 1) {
283         Monom* newMonoms = new Monom[(factor * this->m) + 1];
284         for (int i = 0; i < this->am+1; i++) {
285             newMonoms[i] = this->monoms[i];
286         }
287         delete[] this->monoms;
288         this->monoms = newMonoms;
289         this->m = factor * (this->m) + 1;
290     }
291     else{
292         cout << "Error: factor must be greater than 0";
293     }
294 }
```

This is probably caused in connection by the copy assignment operator of the *Monom* class, because as soon as we disable all methods of the *Monom* class, this warning disappears.

3 The Class - DistPoly.h

This section shows the Header file in which the *DistPoly* class is defined.

```
281 //*****************************************************************
282 // "DistPoly.h"
283 //
284 // is the header, that defines the DistPoly class.
285 //
286 // created by Felix Dressler, 28.04.2022
287 //*****************************************************************
288 #pragma once
289
290 #include<string>
291
292 using namespace std;
293
294 //*****************************************************************
295 // class "DistPoly"
296 //
297 // this class represents polynomials by arrays of monomials and provides
298 // a number of operations with these polynomials.
299 //
300 // Monom ... class that defines monomials
301 // n ... number of variables
302 // vars ... names of the variables
303 // monoms ... array of Monoms
304 // m ... number of potential monoms in this polynomial
305 // am ... actual number of monoms in this polynomial -1
306 //
307 // The member functions are shortly describes directly in the class.
308 // For further documentation see the "DistPoly.h" file
309 //*****************************************************************
310
311 class DistPoly
312 {
313 private:
314   class Monom;
315   int n;
316   string* vars;
317   Monom* monoms;
318   int m;
319   int am;
320
321   //enlarges the polynomial by a given factor greater than or equal two
322   void resize(int factor);
323
324   //gives back the position in which the monom with the exponents exps should be inserted
325   int sort(int* exps, int n, int j);
326
327 public:
328
329   //constructor
330   DistPoly(int n, string* vars);
331
332   //copy constructor, copy assignment operator, destructor
333   DistPoly(DistPoly& p);
334   DistPoly& operator=(DistPoly& p);
335   ~DistPoly();
```

```
336
337   //adds monomials to a polynomial
338   DistPoly& add(int coeff, int* exps);
339
340   //adds polynomials to polynomials
341   DistPoly& add(DistPoly& p);
342
343   //prints a polynomial
344   void println();
345 };
```

4 The Class - DistPoly.cpp

This section shows the .cpp file in which the *DistPoly* class is implemented.

Note: The copy constructor could also be implemented by using the add function.

```
281 //*****************************************************************
282 // "DistPoly.cpp"
283 //
284 // is the cpp file, where the member-functions of the DistPoly Class
285 // are defined.
286 //
287 // created by Felix Dresser, 28.04.2022
288 //*****************************************************************
289
290 #include"DistPoly.h"
291 #include<iostream>
292
293 using namespace std;
294
295 //*****************************************************************
296 // class "Monom"
297 //
298 // The Monom class is a private class of the DistPoly class and
299 // serves as a structure for saving monomials in the DistPoly class.
300 //
301 // coeff ... is the coefficient of a monomial
302 // exps ... is an array of integers, that holds the exponents of the monomial
303 // n ... is the number of variables in a monomial
304 //
305 // member functions ... more description in the comments above them
306 //*****************************************************************
307
308 class DistPoly::Monom
309 {
310 public:
311     int coeff;
312     int* exps;
313     int n;
314
315     //constructors
316     Monom();
317     Monom(int coeff, int* exps, int n);
318
319     //copy constructor, copy assignment operator, destructor
320     Monom& operator=(Monom& m);
321
322     //destructor
323     ~Monom();
324 };
325
326 //*****************************************************************
327 // Method "add(itn coeff, int* exps)"
328 //
329 // is a member function of the class "DistPoly" and adds the monomial
330 // given by its coefficient and its exponent to the current polynomial
331 //
332 // coeff ... ceofficient of the monomial that will be added
333 // exps ... exponents of the monomial that will be added
334 //*****************************************************************
```

```
335
336 DistPoly& DistPoly::add(int coeff, int* exps) {
337     if (coeff !=0) {
338         for (int j = 0; j <= this->m; j++) {
339             int k = 1;
340             //checks where to insert/add the polynomial
341             if (j < this->m) {
342                 k = sort(exps, this->n, j);
343             }
344             if (k == 0) {
345                 if (this->monoms[j].coeff + coeff == 0) {
346                     for (int l = j; l < this->am+1; l++) { //shifts the monoms into the
                            gap to fill it
347                         this->monoms[l] = this->monoms[l + 1];
348                     }
349                     this->am--;
350                 }
351                 else {
352                     this->monoms[j].coeff += coeff;
353                 }
354                 break;
355             }
356             else if (k == -1) {
357
358                 if (this->m >= this->am+1) {
359                     this->resize(2);
360                 }
361
362                 for (int l = this->am+1; l > j; l--) {
363                     this->monoms[l] = this->monoms[l-1];
364                 }
365                 this->monoms[j].coeff = coeff;
366                 delete[] this->monoms[j].exps;
367                 this->monoms[j].exps = new int[this->n];
368                 for (int i = 0; i <this->n; i++)
369                 {
370                     this->monoms[j].exps[i] = exps[i];
371                 }
372                 this->am++;
373                 break;
374             }
375         }
376     }
377
378     return *this;
379 }
380
381 //*****************************************************************
382 // Method "add(DistPoly& p)"
383 //
384 // is a member function of the class "DistPoly" and adds one
385 // polynomial to the polynomial.
386 //
387 // p ... polynomial that should be added
388 //*****************************************************************
389
390 DistPoly& DistPoly::add(DistPoly& p) {
391     if (this->n != p.n) {
392         cout << "Error: the number of variables of two added polynomials is different";
393         exit(2);
```

```
394     }
395     for (int i = 0; i < this->n; i++) {
396         if (this->vars[i] != p.vars[i]) {
397             cout << "Error: the variables of two added polynomials do not match";
398             exit(1);
399         }
400     }
401     if (p.am != 0) {
402         for (int i = 0; i <= p.am; i++){
403             this->add(p.monoms[i].coeff, p.monoms[i].exps);
404         }
405     }
406     return *this;
407 }
408 
409 //*****************************************************************
410 // Method "sort"
411 //
412 // is a private member function of DistPoly used in the add method to determine
413 // the correct place in which the new monomial should be added.
414 //
415 // It takes the exponents of a monomial and gives back
416 //
417 // -> 0 if the given exponents match the exponents of this polynomial.
418 // -> 1 if the given exponents need to be inserted after this polynomial.
419 // -> -1 if the given exponents need to be inserted before this polynomial.
420 //
421 // exps ... exponents to be sorted in
422 // n ... number of variables in this polynomial
423 // j ... the number of the monomial it should check against in the
424 // monomial array of the polynomial
425 //*****************************************************************
426 
427 int DistPoly::sort(int* exps, int n, int j) {
428     for (int i = 0; i < n; i++) {
429         if (this->monoms[j].exps[i] > exps[i]) {
430             return 1;
431         }
432          else if (this->monoms[j].exps[i] < exps[i]) {
433             return -1;
434         }
435     }
436     return 0;
437 }
438 
439 //*****************************************************************
440 // Method "println()"
441 //
442 // is a member function of the "DistPoly" class.
443 // it prints out the given polynomial.
444 //*****************************************************************
445 
446 void DistPoly::println() {
447     if (n == 0 || m==0 || am==0) {
448         cout << "0" << endl;
449     }
450     else {
451         for (int i = 0; i < m; i++) {
452             if (this->monoms[i].coeff != 0) {
453                 cout << this->monoms[i].coeff;
```

```
454                 for (int j = 0; j < n; j++) {
455                     if (this->monoms[i].exps[j] == 1) {
456                         cout << this->vars[j];
457                     }
458                     else if (this->monoms[i].exps[j] != 0) {
459                         cout << this->vars[j];
460                         cout << "^" << this->monoms[i].exps[j];
461                     }
462                 }
463                 if (i < am && this->monoms[i+1].coeff > 0) {
464                     cout << "+";
465                 }
466             }
467         }
468         cout << "\n";
469     }
470 }
471 
472 //*****************************************************************
473 // constructor "DistPoly(int n, string* vars)"
474 //
475 // constructs and initializes Polynomials
476 //*****************************************************************
477 
478 DistPoly::DistPoly(int n, string* vars) {
479     this->n = n;
480     this->vars = new string[n];
481     for (int i = 0; i < n; i++) {
482         this->vars[i] = vars[i];
483     }
484     this->m = 1;
485     this->am = 0;
486     this->monoms = new Monom[m];
487     for (int j = 0; j < m; j++) {
488         this->monoms[j] = *new Monom(0, new int[n] {0}, n);
489     }
490 }
491 
492 //*****************************************************************
493 // copy constructor "DistPoly(DistPoly& p)"
494 //
495 // copy constructor for "DistPoly"
496 //*****************************************************************
497 
498 DistPoly::DistPoly(DistPoly& p) {
499     this->n = p.n;
500     delete[] this->vars;
501     this->vars = new string[n];
502     for (int i = 0; i < n; i++) {
503         this->vars[i] = p.vars[i];
504     }
505     this->m = p.m;
506     this->am = p.am;
507     delete[] this->monoms;
508     this->monoms = new Monom[this->m];
509     for (int i = 0; i < m; i++) {
510         this->monoms[i] = *new Monom(p.monoms[i].coeff, p.monoms[i].exps, p.n);
511     }
512 }
513 
```

```
514 //*************************************************************
515 // copy assignment operator "DistPoly::operator=(DistPoly& p)"
516 //
517 //  copy assignment operator for "DistPoly"
518 //*************************************************************
519
520 DistPoly& DistPoly::operator=(DistPoly& p) {
521     this->n = p.n;
522     delete[] vars;
523     this->vars = new string[n];
524     for (int i = 0; i < n; i++) {
525         this->vars[i] = p.vars[i];
526     }
527     this->m = p.m;
528     this->am = p.am;
529     delete[] monoms;
530     this->monoms = new Monom[this->m];
531     for (int i = 0; i < m; i++) {
532         this->monoms[i] = *new Monom(p.monoms[i].coeff, p.monoms[i].exps, p.n);
533     }
534
535     return *this;
536 }
537
538 //*************************************************************
539 // destructor "~DistPoly()"
540 //
541 // destructor for "DistPoly"
542 //*************************************************************
543
544 DistPoly::~DistPoly() {
545         delete[] this->vars;
546         delete[] this->monoms;
547 }
548
549 //*************************************************************
550 // Method "resize(int factor)"
551 //
552 // is a member function of "DistPoly".
553 // It enlarges the size of the array by a given factor (>1) of polynomials
554 // and copys the old polynomial into it.
555 //
556 // factor ... the factor by which the polynomial should be enlarged
557 //*************************************************************
558
559 void DistPoly::resize(int factor) {
560     if (factor > 1) {
561         Monom* newMonoms = new Monom[(factor * this->m) + 1];
562         for (int i = 0; i < this->am+1; i++) {
563             newMonoms[i] = this->monoms[i];
564         }
565         delete[] this->monoms;
566         this->monoms = newMonoms;
567         this->m = factor * (this->m) + 1;
568     }
569     else{
570         cout << "Error: factor must be greater than 0";
571     }
572 }
573
```

```
574 //*************************************************************
575 // constructor "Monom(int coeff, int* exps, int n)"
576 //
577 // is a constructor for the private member class of "DistPoly"
578 // called "Monom".
579 // It constructs a monomial with the values of:
580 //
581 // coeff ... is the coefficient of the monomial
582 // exps ... is the exponent array
583 // n ... is the number of variables
584 //*************************************************************
585
586 DistPoly::Monom::Monom(int coeff, int* exps, int n) {
587     this->n = n;
588     this->coeff = coeff;
589     this->exps = new int[n]; //creates a new array of exponents, this is in order to have
          seperate pointers and deallocate their respectivve memory later (for DistPoly)
590     for (int i = 0; i < n; i++) {
591         this->exps[i] = exps[i];
592     }
593 }
594
595 //*************************************************************
596 // constructor "Monom()"
597 //
598 // is the empty constructor for the private member class of "DistPoly"
599 // called "Monom".
600 //*************************************************************
601
602 DistPoly::Monom::Monom() {
603     this->n = 1;
604     this->coeff = 0;
605     this->exps = new int[n];
606     for (int i = 0; i < this->n; i++) {
607         this->exps[i] = 0;
608     }
609 }
610
611 //*************************************************************
612 // copy assignment operator "Monom::operator=(Monom& m)"
613 //
614 // is the copy assignment operator for the private member class of "DistPoly"
615 // called "Monom".
616 //*************************************************************
617
618 DistPoly::Monom& DistPoly::Monom::operator=(Monom& m) {
619     this->n = m.n;
620     this->coeff = m.coeff;
621     delete[] exps;
622     this->exps = new int[n]; //creates a new array of exponents, this is in order to have
          seperate pointers and deallocate their respectivve memory later (for DistPoly)
623     for (int i = 0; i < n; i++) {
624         this->exps[i] = m.exps[i];
625     }
626     return *this;
627 }
628
629 //*************************************************************
630 // destructor "~Monom()"
631 //
```

```
632 // is the destructor for the private member class of "DistPoly"
633 // called "Monom".
634 //*****************************************************************
635
636 DistPoly::Monom::~Monom() {
637     delete[] exps;
638 }
```