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.

```
//*************************************************************
// "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.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);
  q.add(11, e4).add(-3, e2).add(2, e6).add(-2, e2);

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

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

  DistPoly r = p;
  r.add(q).add(q);
  p = r;
  p.println();

  return 0;
}
```

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.

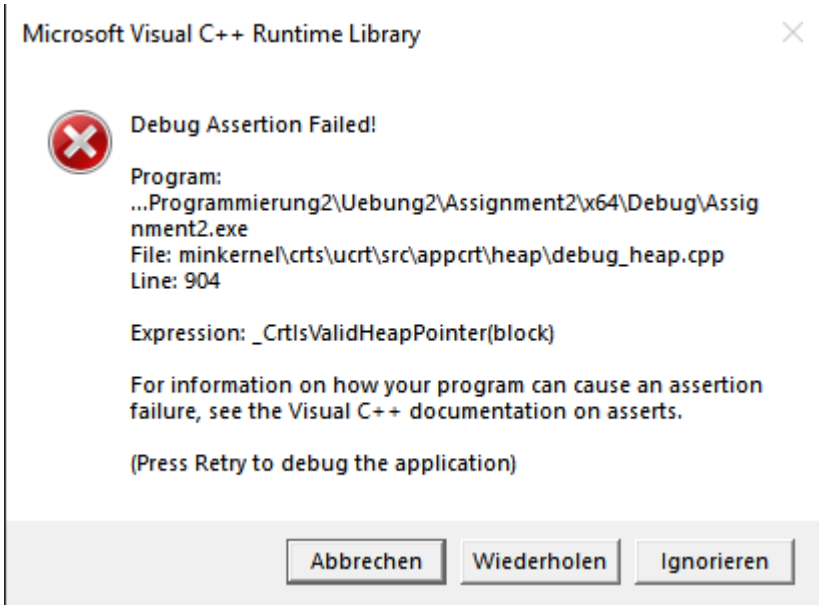
```
Error: the variables of two added polynomials do not match
```

2 Problems

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

2.1 Monom-destructor

The following Error message is displayed, when trying to run the program with the *destructor* of the *Monom* class.

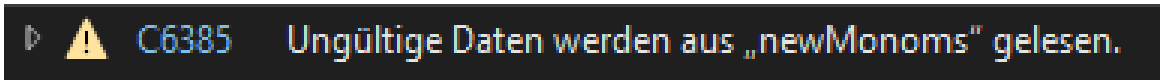


This *destructor* was tried in different versions. Without the check, if the to be deleted exponent is not the NULL pointer, there was the same error message. When debugging it would stop at a predefined library-breakpoint. This is the reason why this destructor was left commented out in the final version, even though this means that there are most probably memory leaks occurring while running the program.

```
375 //DistPoly::Monom::~Monom() {
376 //    if (exps != 0) {
377 //        delete[] exps;
378 //    }
379 //}
```

2.2 warnings

In the *resize* method, line 297 of the *DistPoly.cpp* this warning is displayed:




```
293 void DistPoly::resize(int factor) {
294     if (factor > 1) {
295         Monom* newMonoms = new Monom[(factor * this->m) + 1];
296         for (int i = 0; i < this->am+1; i++) {
297             newMonoms[i] = this->monoms[i];
298         }
```

```
299         delete[] this->monoms;
300         this->monoms = newMonoms;
301         this->m = factor * (this->m) + 1;
302     }
303     else{
304         cout << "Error: factor must be greater than 0";
305     }
306 }
```

This is probably caused 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.

```
293 //*****************************************************************
294 // "DistPoly.h"
295 //
296 // is the header, that defines the DistPoly class.
297 //
298 // created by Felix Dressler, 28.04.2022
299 //*****************************************************************
300 #pragma once
301
302 #include<string>
303
304 using namespace std;
305
306 //*****************************************************************
307 // class "DistPoly"
308 //
309 // this class represents polynomials by arrays of monomials and provides
310 // a number of operations with these polynomials.
311 //
312 // Monom ... class that defines monomials
313 // n ... number of variables
314 // vars ... names of the variables
315 // monoms ... array of Monoms
316 // m ... number of potential monoms in this polynomial
317 // am ... actual number of monoms in this polynomial -1
318 //
319 // The member functions are shortly describes directly in the class.
320 // For further documentation see the "DistPoly.h" file
321 //*****************************************************************
322
323 class DistPoly
324 {
325 private:
326   class Monom;
327   int n;
328   string* vars;
329   Monom* monoms;
330   int m;
331   int am;
332
333   //enlarges the polynomial by a given factor greater than or equal two
334   void resize(int factor);
335
336   //gives back the position in which the monom with the exponents exps should be inserted
337   int sort(int* exps, int n, int j);
338
339 public:
340
341   //constructor
342   DistPoly(int n, string* vars);
343
344   //copy constructor, copy assignment operator, destructor
345   DistPoly(DistPoly& p);
346   DistPoly& operator=(DistPoly& p);
347   ~DistPoly();
```

```
348
349   //adds monomials to a polynomial
350   DistPoly& add(int coeff, int* exps);
351
352   //adds polynomials to polynomials
353   DistPoly& add(DistPoly& p);
354
355   //prints a polynomial
356   void println();
357 };
```

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.

```
293 //*****************************************************************
294 // "DistPoly.cpp"
295 //
296 // is the cpp file, where the member-functions of the DistPoly Class
297 // are defined.
298 //
299 // created by Felix Dresser, 28.04.2022
300 //*****************************************************************
301
302 #include"DistPoly.h"
303 #include<iostream>
304
305 using namespace std;
306
307 //*****************************************************************
308 // class "Monom"
309 //
310 // The Monom class is a private class of the DistPoly class and
311 // serves as a structure for saving monomials in the DistPoly class.
312 //
313 // coeff ... is the coefficient of a monomial
314 // exps ... is an array of integers, that holds the exponents of the monomial
315 // n ... is the number of variables in a monomial
316 //
317 // member functions ... more description in the comments above them
318 //*****************************************************************
319
320 class DistPoly::Monom
321 {
322 public:
323     int coeff;
324     int* exps;
325     int n;
326
327     //constructors
328     Monom();
329
330     //in this project, this constructor was actually never used, it could thus be deleted
331     //it was left in, because it could be usefull for further expanding the class
332     Monom(int coeff, int* exps, int n);
333
334     //copy constructor, copy assignment operator, destructor
335     Monom& operator=(Monom& m);
336
337     //destructor
338     //~Monom();
339 };
340
341 //*****************************************************************
342 // Method "add(itn coeff, int* exps)"
343 //
344 // is a member function of the class "DistPoly" and adds the monomial
345 // given by its coefficient and its exponent to the current polynomial
346 //
```

```
347 // coeff ... ceofficient of the monomial that will be added
348 // exps ... exponents of the monomial that will be added
349 //*************************************************************
350
351 DistPoly& DistPoly::add(int coeff, int* exps) {
352     if (coeff !=0) {
353         for (int j = 0; j <= this->m; j++) {
354             int k = 1;
355             //checks where to insert/add the polynomial
356             if (j < this->m) {
357                 k = sort(exps, this->n, j);
358             }
359             if (k == 0) {
360                 if (this->monoms[j].coeff + coeff == 0) {
361                     for (int l = j; l < this->am+1; l++) { //shifts the monoms into the
                            gap to fill it
362                         this->monoms[l] = this->monoms[l + 1];
363                     }
364                     this->am--;
365                 }
366                 else {
367                     this->monoms[j].coeff += coeff;
368                 }
369                 break;
370             }
371             else if (k == -1) {
372
373                 if (this->m == this->am+1) {
374                     this->resize(2);
375                 }
376
377                 for (int l = this->am + 1; l > j; l--) {
378                     this->monoms[l] = this->monoms[l-1];
379                 }
380                 this->monoms[j].coeff = coeff;
381                 this->monoms[j].exps = exps;
382                 this->am++;
383                 break;
384             }
385         }
386     }
387
388     return *this;
389 }
390
391 //*************************************************************
392 // Method "add(DistPoly& p)"
393 //
394 // is a member function of the class "DistPoly" and adds one
395 // polynomial to the polynomial.
396 //
397 // p ... polynomial that should be added
398 //*************************************************************
399
400 DistPoly& DistPoly::add(DistPoly& p) {
401     if (this->n != p.n) {
402         cout << "Error: the number of variables of two added polynomials is different";
403     }
404     for (int i = 0; i < this->n; i++) {
405         if (this->vars[i] != p.vars[i]) {
```

```
406             cout << "Error: the variables of two added polynomials do not match";
407             exit(1);
408         }
409     }
410     if (p.am != 0) {
411         for (int i = 0; i <= p.am; i++){
412             this->add(p.monoms[i].coeff, p.monoms[i].exps);
413         }
414     }
415 
416     return *this;
417 }
418 
419 //*************************************************************
420 // Method "sort"
421 //
422 // is a private member function of DistPoly used in the add method to determine
423 // the correct place in which the new monomial should be added.
424 //
425 // It takes the exponents of a monomial and gives back
426 //
427 // -> 0 if the given exponents match the exponents of this polynomial.
428 // -> 1 if the given exponents need to be inserted after this polynomial.
429 // -> -1 if the given exponents need to be inserted before this polynomial.
430 //
431 // exps ... exponents to be sorted in
432 // n ... number of variables in this polynomial
433 // j ... the number of the monomial it should check against in the
434 // monomial array of the polynomial
435 //*************************************************************
436 
437 int DistPoly::sort(int* exps, int n, int j) {
438     for (int i = 0; i < n; i++) {
439         if (this->monoms[j].exps[i] > exps[i]) {
440             return 1;
441         }
442          else if (this->monoms[j].exps[i] < exps[i]) {
443             return -1;
444         }
445     }
446     return 0;
447 }
448 
449 //*************************************************************
450 // Method "println()"
451 //
452 // is a member function of the "DistPoly" class.
453 // it prints out the given polynomial.
454 //
455 //*************************************************************
456 
457 void DistPoly::println() {
458     if (n == 0 || m==0 || am==0) {
459         cout << "0" << endl;
460     }
461     else {
462         for (int i = 0; i < m; i++) {
463             if (this->monoms[i].coeff != 0) {
464                 cout << this->monoms[i].coeff;
465 
```

```
466                 for (int j = 0; j < n; j++) {
467                     if (this->monoms[i].exps[j] == 1) {
468                         cout << this->vars[j];
469                     }
470                     else if (this->monoms[i].exps[j] != 0) {
471                         cout << this->vars[j];
472                         cout << "^" << this->monoms[i].exps[j];
473                     }
474                 }
475                 if (i < am && this->monoms[i+1].coeff > 0) {
476                     cout << "+";
477                 }
478             }
479         }
480         cout << "\n";
481     }
482 }
483 
484 //*************************************************************
485 // constructor "DistPoly(int n, string* vars)"
486 //
487 // constructs and initializes Polynomials
488 //*************************************************************
489 
490 DistPoly::DistPoly(int n, string* vars) {
491     this->n = n;
492     this->vars = new string[n];
493     for (int i = 0; i < n; i++) {
494         this->vars[i] = vars[i];
495     }
496     this->m = 1;
497     this->am = 0;
498     this->monoms = new Monom[m];
499     for (int j = 0; j < m; j++) {
500         this->monoms[j].coeff = 0;
501         this->monoms[j].exps = new int[n];
502         this->monoms[j].n = n;
503         for (int i = 0; i < n; i++) {
504             this->monoms[j].exps[i] = 0;
505         }
506     }
507 }
508 
509 //*************************************************************
510 // copy constructor "DistPoly(DistPoly& p)"
511 //
512 // copy constructor for "DistPoly"
513 //*************************************************************
514 
515 DistPoly::DistPoly(DistPoly& p) {
516     this->n = p.n;
517     delete[] this->vars;
518     this->vars = new string[n];
519     for (int i = 0; i < n; i++) {
520         this->vars[i] = p.vars[i];
521     }
522     this->m = p.m;
523     this->am = p.am;
524     delete[] this->monoms;
525     this->monoms = new Monom[this->m];
```

```
526     for (int i = 0; i < m; i++) {
527         this->monoms[i].coeff = p.monoms[i].coeff;
528         this->monoms[i].n = p.n;
529         for (int j = 0; j < p.n; j++) {
530             this->monoms[i].exps[j] = p.monoms[i].exps[j];
531         }
532     }
533 
534 }
535 
536 //****************************************************************
537 // copy assignment operator "DistPoly::operator=(DistPoly& p)"
538 //
539 //  copy assignment operator for "DistPoly"
540 //****************************************************************
541 
542 DistPoly& DistPoly::operator=(DistPoly& p) {
543     this->n = p.n;
544     delete[] vars;
545     this->vars = new string[n];
546     for (int i = 0; i < n; i++) {
547         this->vars[i] = p.vars[i];
548     }
549     this->m = p.m;
550     this->am = p.am;
551     delete[] monoms;
552     this->monoms = new Monom[this->m];
553     for (int i = 0; i < m; i++) {
554         this->monoms[i].coeff = p.monoms[i].coeff;
555         this->monoms[i].n = p.n;
556         for (int j = 0; j < p.n; j++) {
557             this->monoms[i].exps[j] = p.monoms[i].exps[j];
558         }
559     }
560 
561     return *this;
562 }
563 
564 //****************************************************************
565 // destructor "~DistPoly()"
566 //
567 // destructor for "DistPoly"
568 //****************************************************************
569 
570 DistPoly::~DistPoly() {
571         delete[] this->vars;
572         delete[] this->monoms;
573 }
574 
575 //****************************************************************
576 // Method "resize(int factor)"
577 //
578 // is a member function of "DistPoly".
579 // It enlarges the size of the array by a given factor (>1) of polynomials
580 // and copys the old polynomial into it.
581 //
582 // factor ... the factor by which the polynomial should be enlarged
583 //****************************************************************
584 
585 void DistPoly::resize(int factor) {
```

```
586     if (factor > 1) {
587         Monom* newMonoms = new Monom[(factor * this->m) + 1];
588         for (int i = 0; i < this->am+1; i++) {
589             newMonoms[i] = this->monoms[i];
590         }
591         delete[] this->monoms;
592         this->monoms = newMonoms;
593         this->m = factor * (this->m) + 1;
594     }
595     else{
596         cout << "Error: factor must be greater than 0";
597     }
598 }
599 
600 //*****************************************************************
601 // constructor "Monom(int coeff, int* exps, int n)"
602 //
603 // is a constructor for the private member class of "DistPoly"
604 // called "Monom".
605 // It constructs a monomial with the values of:
606 //
607 // coeff ... is the coefficient of the monomial
608 // exps ... is the exponent array
609 // n ... is the number of variables
610 //
611 // this constructor is currently not used
612 //*****************************************************************
613 
614 DistPoly::Monom::Monom(int coeff, int* exps, int n) {
615     this->n = n;
616     this->coeff = coeff;
617     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)
618     for (int i = 0; i < n; i++) {
619         this->exps[i] = exps[i];
620     }
621 
622 }
623 
624 //*****************************************************************
625 // constructor "Monom()"
626 //
627 // is the empty constructor for the private member class of "DistPoly"
628 // called "Monom".
629 //*****************************************************************
630 
631 DistPoly::Monom::Monom() {
632     this->n = 1;
633     this->coeff = 0;
634     this->exps = new int[n];
635     for (int i = 0; i < this->n; i++) {
636         this->exps[i] = 0;
637     }
638 }
639 
640 //*****************************************************************
641 // copy assignment operator "Monom::operator=(Monom& m)"
642 //
643 // is the copy assignment operator for the private member class of "DistPoly"
644 // called "Monom".
```

```
645 //*****************************************************************
646
647 DistPoly::Monom& DistPoly::Monom::operator=(Monom& m) {
648     this->n = m.n;
649     this->coeff = m.coeff;
650     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)
651     for (int i = 0; i < n; i++) {
652         this->exps[i] = m.exps[i];
653     }
654
655     return *this;
656 }
657
658 //*****************************************************************
659 // destructor "~Monom()"
660 //
661 // is the destructor for the private member class of "DistPoly"
662 // called "Monom".
663 //
664 // there are currently problems, involving this destructor!!!
665 //*****************************************************************
666
667 //DistPoly::Monom::~Monom() {
668 //    if (exps != 0) {
669 //        delete[] exps;
670 //    }
671 //}
```