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.

```
//****************************************************************
// "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 vars[2] = { "x","y" };

int main() {

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

  // construct zero polynomial in two variables, then add monomials
  string vars[2] = { "x","y" };
  DistPoly p(2, vars);
  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(2, vars);
  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^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.

```
//*****************************************************************
// "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 vars[1] = { "x" };

int main() {

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

  // construct zero polynomial in two variables, then add monomials
  DistPoly p(1, vars);
  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(1, vars);
  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^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.

```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] = { "x","y","z" };

int main() {

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


  // construct zero polynomial in three variables, then add monomials
  DistPoly p(3, 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 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.

```
//***************************************************************
// "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.

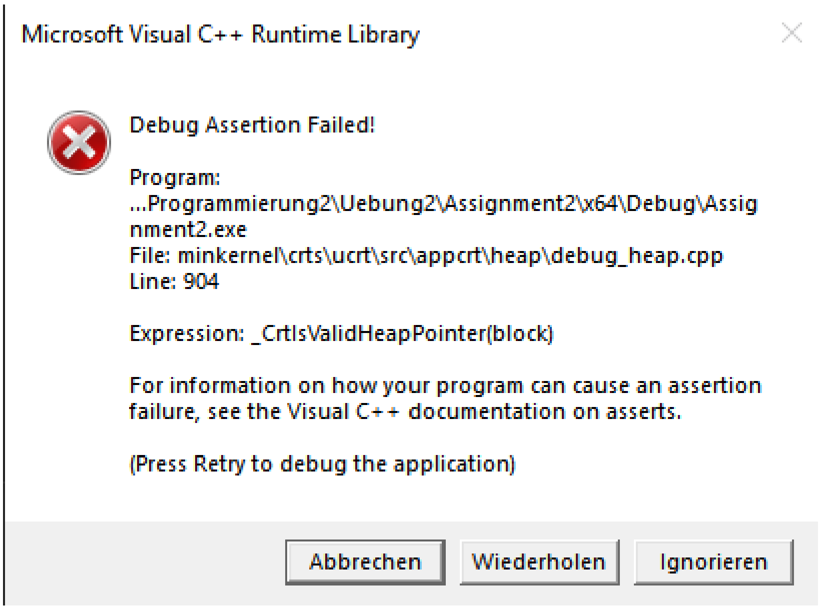
```
Error: the variables of two added polynomials do not match
```

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.



Microsoft Visual C++ Runtime Library

Debug Assertion Failed!

Program:
...Programmierung2\Uebung2\Assignment2\x64\Debug\Assignment2.exe
File: minkernel\crts\ucrt\src\appcrt\heap\debug_heap.cpp
Line: 904

Expression: _CrtIsValidHeapPointer(block)

For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.

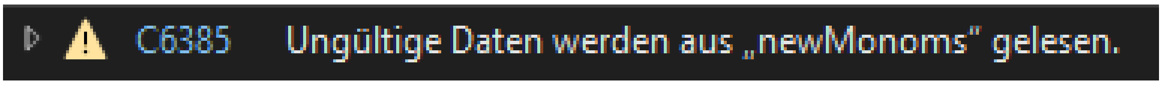
(Press Retry to debug the application)

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.

```
//DistPoly::Monom::~Monom() {
//    if (exps != 0) {
//        delete[] exps;
//    }
//}
```

2.2 warnings

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



C6385 Ungültige Daten werden aus „newMonoms" gelesen.

```
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         }
```

```
        delete[] this->monoms;
        this->monoms = newMonoms;
        this->m = factor * (this->m) + 1;
    }
    else{
        cout << "Error: factor must be greater than 0";
    }
}
```

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.

```
//******************************************************************
// "DistPoly.h"
//
// is the header, that defines the DistPoly class.
//
// created by Felix Dressler, 28.04.2022
//******************************************************************
#pragma once

#include<string>

using namespace std;

//******************************************************************
// class "DistPoly"
//
// this class represents polynomials by arrays of monomials and provides
// a number of operations with these polynomials.
//
// Monom ... class that defines monomials
// n ... number of variables
// vars ... names of the variables
// monoms ... array of Monoms
// m ... number of potential monoms in this polynomial
// am ... actual number of monoms in this polynomial -1
//
// The member functions are shortly describes directly in the class.
// For further documentation see the "DistPoly.h" file
//******************************************************************

class DistPoly
{
private:
  class Monom;
  int n;
  string* vars;
  Monom* monoms;
  int m;
  int am;

  //enlarges the polynomial by a given factor greater than or equal two
  void resize(int factor);

  //gives back the position in which the monom with the exponents exps should be inserted
  int sort(int* exps, int n, int j);

public:

  //constructor
  DistPoly(int n, string* vars);

  //copy constructor, copy assignment operator, destructor
  DistPoly(DistPoly& p);
  DistPoly& operator=(DistPoly& p);
  ~DistPoly();

  //adds monomials to a polynomial
  DistPoly& add(int coeff, int* exps);

  //adds polynomials to polynomials
  DistPoly& add(DistPoly& p);

  //prints a polynomial
  void println();
};
```

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.

```
//******************************************************************
// "DistPoly.cpp"
//
// is the cpp file, where the member-functions of the DistPoly Class
// are defined.
//
// created by Felix Dresser, 28.04.2022
//******************************************************************

#include"DistPoly.h"
#include<iostream>

using namespace std;

//******************************************************************
// class "Monom"
//
// The Monom class is a private class of the DistPoly class and
// serves as a structure for saving monomials in the DistPoly class.
//
// coeff ... is the coefficient of a monomial
// exps ... is an array of integers, that holds the exponents of the monomial
// n ... is the number of variables in a monomial
//
// member functions ... more description in the comments above them
//******************************************************************

class DistPoly::Monom
{
public:
    int coeff;
    int* exps;
    int n;

    //constructors
    Monom();

    //in this project, this constructor was actually never used, it could thus be deleted
    //it was left in, because it could be usefull for further expanding the class
    Monom(int coeff, int* exps, int n);

    //copy constructor, copy assignment operator, destructor
    Monom& operator=(Monom& m);

    //destructor
    //~Monom();
};

//******************************************************************
// Method "add(itn coeff, int* exps)"
//
// is a member function of the class "DistPoly" and adds the monomial
// given by its coefficient and its exponent to the current polynomial
//
```

```
// coeff ... ceofficient of the monomial that will be added
// exps ... exponents of the monomial that will be added
//************************************************************

DistPoly& DistPoly::add(int coeff, int* exps) {
    if (coeff !=0) {
        for (int j = 0; j <= this->m; j++) {
            int k = 1;
            //checks where to insert/add the polynomial
            if (j < this->m) {
                k = sort(exps, this->n, j);
            }
            if (k == 0) {
                if (this->monoms[j].coeff + coeff == 0) {
                    for (int l = j; l < this->am+1; l++) { //shifts the monoms into the gap to fill it
                        this->monoms[l] = this->monoms[l + 1];
                    }
                    this->am--;
                }
                else {
                    this->monoms[j].coeff += coeff;
                }
                break;
            }
            else if (k == -1) {

                if (this->m == this->am+1) {
                    this->resize(2);
                }

                for (int l = this->am + 1; l > j; l--) {
                    this->monoms[l] = this->monoms[l-1];
                }
                this->monoms[j].coeff = coeff;
                this->monoms[j].exps = exps;
                this->am++;
                break;
            }
        }
    }

    return *this;
}

//************************************************************
// Method "add(DistPoly& p)"
//
// is a member function of the class "DistPoly" and adds one
// polynomial to the polynomial.
//
// p ... polynomial that should be added
//************************************************************

DistPoly& DistPoly::add(DistPoly& p) {
    if (this->n != p.n) {
        cout << "Error: the number of variables of two added polynomials is different";
    }
    for (int i = 0; i < this->n; i++) {
        if (this->vars[i] != p.vars[i]) {
            cout << "Error: the variables of two added polynomials do not match";
            exit(1);
        }
    }
    if (p.am != 0) {
        for (int i = 0; i <= p.am; i++){
            this->add(p.monoms[i].coeff, p.monoms[i].exps);
        }
    }

    return *this;
}

//******************************************************************
// Method "sort"
//
// is a private member function of DistPoly used in the add method to determine
// the correct place in which the new monomial should be added.
//
// It takes the exponents of a monomial and gives back
//
// -> 0 if the given exponents match the exponents of this polynomial.
// -> 1 if the given exponents need to be inserted after this polynomial.
// -> -1 if the given exponents need to be inserted before this polynomial.
//
// exps ... exponents to be sorted in
// n ... number of variables in this polynomial
// j ... the number of the monomial it should check against in the
// monomial array of the polynomial
//******************************************************************

int DistPoly::sort(int* exps, int n, int j) {
    for (int i = 0; i < n; i++) {
        if (this->monoms[j].exps[i] > exps[i]) {
            return 1;
        }
         else if (this->monoms[j].exps[i] < exps[i]) {
            return -1;
        }
    }
    return 0;
}

//******************************************************************
// Method "println()"
//
// is a member function of the "DistPoly" class.
// it prints out the given polynomial.
//
//******************************************************************

void DistPoly::println() {
    if (n == 0 || m==0 || am==0) {
        cout << "0" << endl;
    }
    else {
        for (int i = 0; i < m; i++) {
            if (this->monoms[i].coeff != 0) {
                cout << this->monoms[i].coeff;

                for (int j = 0; j < n; j++) {
                    if (this->monoms[i].exps[j] == 1) {
                        cout << this->vars[j];
                    }
                    else if (this->monoms[i].exps[j] != 0) {
                        cout << this->vars[j];
                        cout << "^" << this->monoms[i].exps[j];
                    }
                }
                if (i < am && this->monoms[i+1].coeff > 0) {
                    cout << "+";
                }
            }
        }
        cout << "\n";
    }
}

//******************************************************************
// constructor "DistPoly(int n, string* vars)"
//
// constructs and initializes Polynomials
//******************************************************************

DistPoly::DistPoly(int n, string* vars) {
    this->n = n;
    this->vars = new string[n];
    for (int i = 0; i < n; i++) {
        this->vars[i] = vars[i];
    }
    this->m = 1;
    this->am = 0;
    this->monoms = new Monom[m];
    for (int j = 0; j < m; j++) {
        this->monoms[j].coeff = 0;
        this->monoms[j].exps = new int[n];
        this->monoms[j].n = n;
        for (int i = 0; i < n; i++) {
            this->monoms[j].exps[i] = 0;
        }
    }
}

//******************************************************************
// copy constructor "DistPoly(DistPoly& p)"
//
// copy constructor for "DistPoly"
//******************************************************************

DistPoly::DistPoly(DistPoly& p) {
    this->n = p.n;
    delete[] this->vars;
    this->vars = new string[n];
    for (int i = 0; i < n; i++) {
        this->vars[i] = p.vars[i];
    }
    this->m = p.m;
    this->am = p.am;
    delete[] this->monoms;
    this->monoms = new Monom[this->m];
    for (int i = 0; i < m; i++) {
        this->monoms[i].coeff = p.monoms[i].coeff;
        this->monoms[i].n = p.n;
        for (int j = 0; j < p.n; j++) {
            this->monoms[i].exps[j] = p.monoms[i].exps[j];
        }
    }

}

//******************************************************************
// copy assignment operator "DistPoly::operator=(DistPoly& p)"
//
//  copy assignment operator for "DistPoly"
//******************************************************************

DistPoly& DistPoly::operator=(DistPoly& p) {
    this->n = p.n;
    delete[] vars;
    this->vars = new string[n];
    for (int i = 0; i < n; i++) {
        this->vars[i] = p.vars[i];
    }
    this->m = p.m;
    this->am = p.am;
    delete[] monoms;
    this->monoms = new Monom[this->m];
    for (int i = 0; i < m; i++) {
        this->monoms[i].coeff = p.monoms[i].coeff;
        this->monoms[i].n = p.n;
        for (int j = 0; j < p.n; j++) {
            this->monoms[i].exps[j] = p.monoms[i].exps[j];
        }
    }

    return *this;
}

//******************************************************************
// destructor "~DistPoly()"
//
// destructor for "DistPoly"
//******************************************************************

DistPoly::~DistPoly() {
        delete[] this->vars;
        delete[] this->monoms;
}

//******************************************************************
// Method "resize(int factor)"
//
// is a member function of "DistPoly".
// It enlarges the size of the array by a given factor (>1) of polynomials
// and copys the old polynomial into it.
//
// factor ... the factor by which the polynomial should be enlarged
//******************************************************************

void DistPoly::resize(int factor) {
    if (factor > 1) {
        Monom* newMonoms = new Monom[(factor * this->m) + 1];
        for (int i = 0; i < this->am+1; i++) {
            newMonoms[i] = this->monoms[i];
        }
        delete[] this->monoms;
        this->monoms = newMonoms;
        this->m = factor * (this->m) + 1;
    }
    else{
        cout << "Error: factor must be greater than 0";
    }
}

//*****************************************************************
// constructor "Monom(int coeff, int* exps, int n)"
//
// is a constructor for the private member class of "DistPoly"
// called "Monom".
// It constructs a monomial with the values of:
//
// coeff ... is the coefficient of the monomial
// exps ... is the exponent array
// n ... is the number of variables
//
// this constructor is currently not used
//*****************************************************************

DistPoly::Monom::Monom(int coeff, int* exps, int n) {
    this->n = n;
    this->coeff = coeff;
    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)
    for (int i = 0; i < n; i++) {
        this->exps[i] = exps[i];
    }

}

//*****************************************************************
// constructor "Monom()"
//
// is the empty constructor for the private member class of "DistPoly"
// called "Monom".
//*****************************************************************

DistPoly::Monom::Monom() {
    this->n = 1;
    this->coeff = 0;
    this->exps = new int[n];
    for (int i = 0; i < this->n; i++) {
        this->exps[i] = 0;
    }
}

//*****************************************************************
// copy assignment operator "Monom::operator=(Monom& m)"
//
// is the copy assignment operator for the private member class of "DistPoly"
// called "Monom".
```

```
//******************************************************************

DistPoly::Monom& DistPoly::Monom::operator=(Monom& m) {
    this->n = m.n;
    this->coeff = m.coeff;
    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)
    for (int i = 0; i < n; i++) {
        this->exps[i] = m.exps[i];
    }

    return *this;
}

//******************************************************************
// destructor "~Monom()"
//
// is the destructor for the private member class of "DistPoly"
// called "Monom".
//
// there are currently problems, involving this destructor!!!
//******************************************************************

//DistPoly::Monom::~Monom() {
//    if (exps != 0) {
//        delete[] exps;
//    }
//}
```