Programming 2 - Assignment 4

Felix Dreßler (k12105003)
email FelixDressler01@gmail.com

May 24, 2022

1 The Program

1.1 The Program - Ring

```cpp
#pragma once
#include<string>

using namespace std;

class Ring {
public:
  // destructor
  virtual ~Ring() {}

  // a heap-allocated duplicate of this element
  virtual Ring* clone() = 0;

  // the string representation of this element
  virtual string str() = 0;

  // the constant of the type of this element and the inverse of this element
  virtual Ring* zero() = 0;
  virtual Ring* operator-() = 0;

  // sum and product of this element and c
  virtual Ring* operator+(Ring* c) = 0;
  virtual Ring* operator*(Ring* c) = 0;

  // comparison function
  virtual bool operator==(Ring* c) = 0;
};
```

1.2 The Program - Integer

```cpp
#pragma once
#include"Ring.h"

class Integer : public Ring {
private:
  int n;
public:
  // integer with value n (default 0)
  Integer(int n = 0);

  // destructor
  virtual ~Integer() {

  }

  // a heap-allocated duplicate of this element
  virtual Ring* clone();

  // the string representation of this element
  virtual string str();

  // the constant of the type of this element and the inverse of this element
  virtual Ring* zero();
  virtual Ring* operator-();

  // sum and product of this element and c
  virtual Ring* operator+(Ring* c);
  virtual Ring* operator*(Ring* c);

  // comparison function
  virtual bool operator==(Ring* c);
};
```

```cpp
#include"Integer.h"
#include<iostream>

using namespace std;

// integer with value n (default 0)
Integer::Integer(int n) {
    this->n = n;
}

// a heap-allocated duplicate of this element
Ring* Integer::clone() {
    Integer* c = new Integer(this->n);

    return c;
}

// the string representation of this element
string Integer::str() {

    return to_string(this->n);;
}

// the constant of the type of this element and the inverse of this element
Ring* Integer::zero() {

    return new Integer(0);
}

Ring* Integer::operator-() {

    return new Integer(-(this->n));
}

// sum and product of this element and c
Ring* Integer::operator+(Ring* c) {
    Integer* x = dynamic_cast<Integer*>(c);

    if (x == 0) {
        cout << "Error: Addition with incompatible Elements performed" << endl;
        exit(1);
    }

    this->n += x->n;

    return this;//like this
}

```

```
Ring* Integer::operator*(Ring* c) {
    Integer* x = dynamic_cast<Integer*>(c);

    if (x == 0) {
        cout << "Error: Multiplikation with incompatible Elements performed" << endl;
        exit(2);
    }

    return new Integer(this->n * x->n);
}

// comparison function
bool Integer::operator==(Ring* c) {
    Integer* x = dynamic_cast<Integer*>(c);

    if (x == 0) {
        cout << "Error: Comparison with incompatible Elements performed" << endl;
        exit(3);
    }

    if (this->n == x->n) {
        return true;
    }
    else {
        return false;
    }

}
```

1.3 The Program - RecPoly

```
#pragma once
#include"Ring.h"

class RecPoly : public Ring {
private:
  Ring** coeff;
  int n;
  string var;

public:
  // polynomial with n>=0 coefficients and given variable name
  RecPoly(string var, int n, Ring** coeffs);
  // copy constructor, copy assignment operator, destructor
  RecPoly(RecPoly& p);
  RecPoly& operator=(RecPoly& p);
  virtual ~RecPoly();

  // a heap-allocated duplicate of this element
  virtual Ring* clone();

  // the string representation of this element
  virtual string str();

  // the constant of the type of this element and the inverse of this element
  virtual Ring* zero();
  virtual Ring* operator-();

  // sum and product of this element and c
  virtual Ring* operator+(Ring* c);
  virtual Ring* operator*(Ring* c);

  // comparison function
  virtual bool operator==(Ring* c);
};
```

```
#include"RecPoly.h"
#include"Integer.h"

#include<iostream>

using namespace std;


// polynomial with n>=0 coefficients and given variable name
RecPoly::RecPoly(string var, int n, Ring** coeffs) {
    this->var = var;
    this->n = n;
    this->coeff = new Ring*[n];
    for (int i = 0; i < n; i++) {
        coeff[i] = coeffs[i]->clone();//clone to make sure only we have control over the
            array
    }
}

// copy constructor, copy assignment operator, destructor
RecPoly::RecPoly(RecPoly& p) {
    this->var = p.var;
    this->n = p.n;

    this->coeff = new Ring * [n];
    for (int i = 0; i < n; i++) {
        coeff[i] = p.coeff[i]->clone();
    }
}

RecPoly& RecPoly::operator=(RecPoly& p) {
    this->var = p.var;
    this->n = p.n;

    this->coeff = new Ring * [n];
    for (int i = 0; i < n; i++) {
        coeff[i] = p.coeff[i]->clone();
    }

    return *this;
}

RecPoly::~RecPoly() {
    for (int i = 0; i < this->n;i++) {
        delete coeff[i];
    }
    delete[] coeff;
}

// a heap-allocated duplicate of this element
Ring* RecPoly::clone() {
```

```cpp

    return new RecPoly(*this);
}

// the string representation of this element
string RecPoly::str() {
    string str = "";
    if (n == 0) {
        str = "0";
    }
    else {
        str += "(";
        for (int i = 0; i < n; i++) {
            if (!(coeff[i]->operator==(coeff[i]->zero()))) {
                str += coeff[i]->str() + "*" + var + "^" + to_string(i);
                    if (i < n - 1) {
                        str += "+";
                    }
            }
        }
        str += ")";
    }

    return str;
}

// the constant of the type of this element and the inverse of this element
Ring* RecPoly::zero() {
    return new RecPoly(this->var, 0, {});
}

Ring* RecPoly::operator-() {

    for (int i = 0; i < this->n; i++) {
        this->coeff[i] = this->coeff[i]->operator-();
    }

    return this;
}

// sum and product of this element and c
Ring* RecPoly::operator+(Ring* c) {

    RecPoly* x = dynamic_cast<RecPoly*>(c);

    if (x == 0) {
        cout << "Error: Addition with incompatible Elements performed" << endl;
        exit(3);
    }
    else {
        int n_temp = 0;

        if (this->n >= x->n) {
            n_temp = this->n;
        }
        else {
            n_temp = x->n;
        }

        Ring** temp = new Ring * [n_temp];

        if (this->n == 0) {
            for (int i = 0; i < x->n; i++) {
                temp[i] = x->coeff[i]->clone();
            }

            RecPoly* add = new RecPoly(this->var, x->n, temp);

            //delete

            return add;
        }
        else {

            for (int i = 0; i < this->n && i < x->n; i++) {
                temp[i] = this->coeff[i]->operator+(x->coeff[i]);
            }


            if (this->n > x->n) {
                for (int i = x->n; i < this->n; i++) {
                    temp[i] = this->coeff[i]->operator+(x->coeff[i]);//why???
                }
            }
            else if(this->n < x->n) {
                for (int i = this->n; i < x->n; i++) {
                    temp[i] = this->coeff[i]->operator+(x->coeff[i]);
                }
            }
            //for (int i = 0; i < this->n; i++) {
            //    temp[i] = c->operator+(this->coeff[i]);//important
            //}

            RecPoly* add = new RecPoly(this->var, this->n, temp);

            for (int i = 0; i < n_temp; i++) {
                delete temp[i];
            }
            delete[] temp;

            return add;
        }
    }


}

Ring* RecPoly::operator*(Ring* c) {

    RecPoly* x = dynamic_cast<RecPoly*>(c);

    if (x == 0) {
        cout << "Error: Multiplication with incompatible Elements performed" << endl;
        exit(4);
    }
    if (this->var != x->var) {
        cout << "Error: Multiplication with incompatible Polynomials performed (wrong
           variables)" << endl;
        exit(4);
    }
```

```cpp

    else {
        if (this->n == 0 || x->n == 0) {
            return new RecPoly(this->var,0,{});
        }
        else{
            Ring** temp = new Ring*[this->n + x->n];

            for (int i = 0; i < this->n + x->n; i++) {
                //temp[i] = 0;
                for (int j = 0; j < i; j++) {
                    temp[i] = *temp[i] + *this->coeff[i] * x->coeff[i];
                }//->operator ->operator
            }

            RecPoly* mult = new RecPoly(this->var, this->n, temp);

            //for (int i = 0; i < this->n; i++) {
            //    delete temp[i];
            //}
            //delete[] temp;

            return mult;
        }
    }

}

// comparison function
bool RecPoly::operator==(Ring* c) {

    for (int i = 0; i < this->n; i++) {
        if (this->coeff[i] != c) {
            return false;
        }
    }
    return true;
}
```

1.4 The Program - Main

```cpp
#include "Integer.h"
#include "RecPoly.h"
#include <iostream>

using namespace std;

int main() {
  Ring* c[] = { new Integer(-5), new Integer(2), new Integer(0), new Integer(-3) };

  RecPoly* p = new RecPoly("x", 4, c); // p = -3x^3 + 2x - 5

  cout << p->str() << endl;

  RecPoly* q = //new RecPoly(*p); //= // q = p+p = -6x^3 + 4x - 10
    dynamic_cast<RecPoly*>(p->operator+(p));
    cout << q->str() << endl;
  RecPoly* r = // r = p*q
```

```
    dynamic_cast<RecPoly*>(p->operator*(q));
  cout << r->str() << endl;



  return 0;
}
```