A data structure for polynomial manipulations. Labwork

December 29, 2019

Deadline: December 13, 2019.

The purpose of this lab is to develop a data structure to work with univariate polynomials with floating-point coefficients.

A univariate polynomial with floating-point coefficients is

$$p = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$$

where x is the indeterminate of the polynomial p, and $a_0, a_1, \ldots, a_n \in \mathbb{R}$ are the coefficients of p. We assume that $a_n \neq 0$. The set of all polynomials of this kind is $\mathbb{R}[x]$.

We want design a data structure for such polynomials, with the following operations:

1. $\deg(p)$ returns the degree of polynomial p. If $p = a_0+a_1\cdot x+\ldots+a_n\cdot x^n$ with $a_n \neq 0$ then $\deg(p) = n$.

2. $\mathrm{lc}(p)$ returns the leading coefficient of polynomial p.
   If $p = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$ with $a_n \neq 0$ then $\mathrm{lc}(p) = a_n$.

3. $\mathrm{coef}(p, i)$ returns the coefficient a_i of x^i in p.

4. $\mathrm{psum}(p, q)$ returns the sum of polynomials p and q.

5. $\mathrm{pprod}(p, q)$ returns the product of polynomials p and p.

6. $\mathrm{pquot}(p, q)$ returns the quotient of dividing p by q.

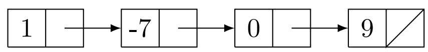
7. $\mathrm{prem}(p, q)$ returns the remainder of dividing p by q.

8. $\mathrm{peval}(p, c)$ returns the value $p(c)$ for some $c \in \mathbb{R}$. If $p = a_0 + a_1 \cdot x + \ldots + a_n \cdot x^n$ then $p(c) = a_0 + a_1 \cdot c + \ldots + a_n \cdot c^n$.

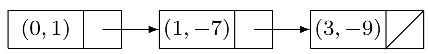
In Lecture 10, we mentioned two representations of univariate polynomials:

- The **dense representation**, which stores all coefficients $a_0, a_1, \ldots, a_n$ in a simply linked list.
- The **sparse representation**, which stores only the nonzero coefficients a_i together with the power i of x in p.

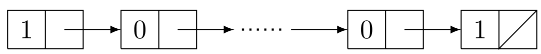
For example, $p = 1 - 7 \cdot x + 9 \cdot x^3$ has the dense list representation



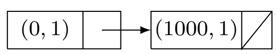
and the sparse list representation



$g = 1 + x^{1000}$ has the dense representation



and the sparse representation



A suitable way to implement the nodes of the dense list representation of a univariate polynomial is with the C++ class

```
struct DRepr {
   float coeff;
   DRepr* next;
 };
```

To perform the polynomial operations mentioned before, consider implementing the static methods of the `C++` class

```
class PolyOps {
  typedef DRepr* Poly;
  static int deg(Poly p);
  static float lc(Poly p);
  static Poly psum(Poly p,Poly q);
  static Poly pprod(Poly p,Poly q);
  static Poly pquot(Poly p,Poly q);
  static Poly prem(Poly p,Poly q);
  static float peval(Poly p,float c);
  static string toString(Poly p);
}
```

The last method is intended to return a string representation of the polynomial represented by `p`, and can be defined as follows:

```
string toString(Poly p) {
if (p==nullptr) return "0";
ostringstream ostr;
ostr << "";
int i=0;
while (p!=nullptr) {
    float c=p->coeff;
    if(i==0) ostr<<c;
    else
      if(c!=0) {
        ostr<<(c>0)?'+':'-';
        c=abs(c);
        if(c!=1) ostr<<c<<'*';
        ostr<<'x';
        if(i>1) ostr<<'^'<<i;
      }
    i++;
    p=p->next;
  }
  return ostr.str();
}
```

For example, the string representation of $1 - 7\,x + 9\,x^3 + x^4 - x^5$ is

```
1-7*x+9*x^3+x^4-x^5
```

Labwork

Implement the missing methods of class `PolyOps` and write a `C++` program that behaves as follows:

- It asks the user to type the coefficients of a polynomial p, on one line, separated by spaces:

  `type the coefficients of p:`
  $a_n \;\; a_{n-1} \;\; \ldots \;\; a_1 \;\; a_0$

  and creates the dense list representation of the polynomial
  $p = a_0 + a_1\,x + \ldots + a_{n-1}\,x^{n-1} + a_n\,x^n$

- It asks the user to type the coefficients of a polynomial q, on one line, separated by spaces:

  ```
  type the coefficients of q:
  ```
  $b_m \ b_{m-1} \ \ldots \ b_1 \ b_0$

  and creates the dense list representation of the polynomial
  $q = b_0 + b_1\,x + \ldots + b_{m-1}x^{m-1} + b_m\,x^m$

- It asks the user to type a value v for a floating point variable `c`:

  `type the value of c:` v

- It computes p(`c`), and the sum, product, quotient and remainder of p and q, and shows them to the user:

  ```
  The value of p(c) is ...
  psum(p,q) = ...
  pprod(p,q) = ...
  pquot(p,q) = ...
  prem(p,q) = ...
  ```

Remark: The following function provides easy way to create the dense list of a polynomial from the string of its coefficients:

```
Poly getPoly(string& coeff_list) {
   float c;
   Poly p = nullptr;
   istringstream iss(coef_list);
   while (iss >> c)
      p=new DRepr(c,p);
   return p;
}
```

(Note: in this implementation, we assumed that class `DRepr` was extended with a suitable constructor `DRepr(float,DRepr*)`)