EE231000 Introduction to Programming

Lab12. Polynomials

Due: May 24, 2014

The linked list is an ideal structure to store polynomials. An n-th degree polynomial has the following form.

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \tag{12.1}$$

And it can be stored using the following structure for the nodes of a linked list.

```
typedef struct sPoly {
    int degree;          // the degree of the node
    double coef;         // the coefficient of the node
    struct sPoly *next;  // pointer to the next node
} POLY;
```

Using the structure defined above to represent polynomials, please write the following functions.

`POLY *oneTerm(int degree,double coef);`
This function creates a 1-term polynomial of the form $\texttt{coef} \times x^{\texttt{degree}}$ and returns the new polynomial.

`POLY *add(POLY *p1,POLY *p2);`
This function adds two polynomials `p1` and `p2` to form a new polynomial and return the new polynomial.

`POLY *sub(POLY *p1,POLY *p2);`
This function subtract polynomial `p2` from `p1` to form a new polynomial and return the new polynomial.

`POLY *mply(POLY *p1,POLY *p2);`
This function multiplies two polynomials `p1` and `p2` to form a new polynomial and return the new polynomial.

`void print(POLY *p1);`
This function prints out the polynomial `p1` in human readable form. See the example output given below for more details.

`void release(POLY *p1);`
This function releases all nodes of the polynomial `p1` and returns them to the *heap space*.

To facilitate polynomial manipulations, all polynomials in this lab should be arranged in degree-descending order. Thus, the first node of a degree n polynomial starts with $a_n x^n$ and, thus, the first node has `degree`$= n$ and `coef`$= a_n$. And to take advantage of the linked list's dynamic property, no term of zero coefficient should be stored. Of course, your functions should not have *memory leak* problem.

Using the functions given above, your `main` function implements the following pseudo code.

Let `X` be a one-term polynomial and `X`= x,
and `One` be a one-term polynomial with `One`=1.
Polynomial A =`X`+`One`,
Polynomial $A_2 = A \times A$,
Polynomial $A_3 = A_2 \times A$,
Polynomial $A_4 = A_3 \times A$,
Polynomial $A_5 = A_4 \times A$,
`printf("A = ");`
`print(`A`)`,
`printf("A2 = ");`
`print(`A_2`)`,
Polynomial B =`X`−`One`,
Polynomial $B_2 = B \times B$,
Polynomial $B_3 = B_2 \times B$,
Polynomial $B_4 = B_3 \times B$,
Polynomial $B_5 = B_4 \times B$,
Polynomial $C = A + B$,
Polynomial $C_2 = A_2 \times B_2$,
Polynomial $C_3 = A_3 \times B_3$,
Polynomial $C_4 = A_4 \times B_4$,
Polynomial $C_5 = A_5 \times B_5$,
`printf("C = ");`
`print(`C`)`,
`printf("C2 = ");`
`print(`C_2`)`,
`printf("C3 = ");`
`print(`C_3`)`,
`printf("C4 = ");`
`print(`C_4`)`,
`printf("C5 = ");`
`print(`C_5`)`.

And the first 4 lines of output look like:

```
A = x +1
A2 = x^2 +2 x +1
C = 2 x
C2 = x^4 -2 x^2 +1
```

Note that if a coefficient is one, the it would not be printed unless it is a constant term. The degree of 1 is also not printed. Please make sure your program output follows the example exactly.

Notes.

1. Create a directory `lab12` and use it as the working directory.
2. Name your program source file as `lab12.c`.
3. The first few lines of your program should be comments as the following.

   ```
   /* EE231002 Lab12. Polynomials
   ```

```
   ID, Name
   Date:
*/
```

4. After you finish verifying your program, you can submit your source code by

   ```
   $ ~ee231002/bin/submit lab12 lab12.c
   ```

   If you see a "submitted successfully" message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   ```
   $ ~ee231002/bin/subrec
   ```

   It will show the last few submission records.

5. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.