lab12

```
$ gcc lab12.c
$ ./a.out
A = x +1
A2 = x^2 +2 x +1
C = 2 x
C2 = x^4 -2 x^2 +1
C3 = x^6 -3 x^4 +3 x^2 -1
C4 = x^8 -4 x^6 +6 x^4 -4 x^2 +1
C5 = x^10 -5 x^8 +10 x^6 -10 x^4 +5 x^2 -1
CPU time: 0.0056277 sec
score: 75
o. [Output] Program output is correct, good.
o. [Format] Program format can be improved
o. [Coding] lab12.c spelling errors: addres(1), algorthim(1), evry(1), realease(1),
tansform(1)
o. [add] function can be improved.
o. [sub] function can be improved.
o. [Memory] leak in multiplication function.
o. [Terms] with zero coefficient should not be stored.
```

lab12.c

```c
// EE2310 lab12. Polynomials
// 109061217, 林峻霆
// Date: 2020/12/22

#include <stdio.h>
#include <stdlib.h>

typedef struct sPoly {   // a struct store a term's degree, coef and next term
    int degree;          // term's degree
    double coef;         // terms coefficient
    struct sPoly *next;  // next term's address
} POLY;

POLY *oneTerm(int degree, double coef);// function: build a one term polynomial
POLY *oneTerm(int degree, double coef); // function: build a one term polynomial
POLY *add(POLY *p1, POLY *p2);         // function: add 2 polynomial together
POLY *sub(POLY *p1, POLY *p2);         // function: subtraction of 2 polynomial
POLY *mply(POLY *p1, POLY *p2);        // function: multiply two polynomial
void print(POLY *p1);                  // function: print the polynomial
void release(POLY *p1);                // function: release polynomial's memory

int main()
int main(void)
{
    POLY *X = oneTerm(1, 1);           // X = x
    POLY *ONE = oneTerm(0, 1);         // ONE = 1

    POLY *A = add(X, ONE);             // A = X + ONE
    POLY *A2 = mply(A, A);             // A2 = A * A
    POLY *A3 = mply(A, A2);            // A3 = A2 * A
    POLY *A4 = mply(A, A3);            // A4 = A3 * A
    POLY *A5 = mply(A, A4);            // A5 = A4 * A

    POLY *B = sub(X, ONE);             // B = X - ONE
    POLY *B2 = mply(B, B);             // B2 = B * B
    POLY *B3 = mply(B, B2);            // B3 = B2 * B
    POLY *B4 = mply(B, B3);            // B4 = B3 * B
    POLY *B5 = mply(B, B4);            // B5 = B4 * B

    POLY *C = add(A, B);               // C = A + B
```

```
39      POLY *C2 = mply(A2, B2);            // C2 = A2 * B2
40      POLY *C3 = mply(A3, B3);            // C3 = A3 * B3
41      POLY *C4 = mply(A4, B4);            // C4 = A4 * B4
42      POLY *C5 = mply(A5, B5);            // C5 = A5 * B5
43
44      printf("A =");                      // print the result
45      print(A);
46      printf("A2 =");
47      print(A2);
48      printf("C =");
49      print(C);
50      printf("C2 =");
51      print(C2);
52      printf("C3 =");
53      print(C3);
54      printf("C4 =");
55      print(C4);
56      printf("C5 =");
57      print(C5);
58
59      return 0;                           // end the program
60  }
61
62  POLY *oneTerm(int degree, double coef)
63      // build a one term polynomial with two inputs: degree, coef and one output
64      //  Inputs:
65      //      degree: term's degree
66      //      coef: term's coefficient
67      //  Outputs:
68      //      new_node: pointer of the new build term
69  {
70      POLY *new_node = malloc(sizeof(POLY)); // assign memory for term
71
72      new_node->degree = degree;              // assign degree
73      new_node->coef = coef;                  // assign coefficient
74      new_node->next = NULL;                  // pointer to next point to NULL
75
76      return new_node;                        // return the term
77  }
78
79  POLY *add(POLY *p1, POLY *p2)
```

```
80      // add two input polynomial together and return the result
81  {
82      POLY *front;      // pointer to the first term of result polynomial
83      POLY *now;        // pointer to the current term of result polynomial
84      POLY *cur1 = p1;  // pointer to the current term of p1 polynomial
85      POLY *cur2 = p2;  // pointer to the current term of p2 polynomial
86
87      // check the condition and then compute the first term of result polynomial
88      if (cur1->degree > cur2->degree) {
89          front = oneTerm(cur1->degree, cur1->coef);
90          cur1 = cur1->next;
91      }
92      else if(cur1->degree < cur2->degree) {
        else if (cur1->degree < cur2->degree) {
93          front = oneTerm(cur2->degree, cur2->coef);
94          cur2 = cur2->next;
95      }
96      else {
97          front = oneTerm(cur1->degree, cur1->coef + cur2->coef);
98          cur1 = cur1->next;
99          cur2 = cur2->next;
100     }
101     now = front;      // point now to front
102
103     // check condition and compute the rest of terms(same method as the above)
104     while (cur1 && cur2) {
105         if (cur1->degree > cur2->degree) {
106             now->next = oneTerm(cur1->degree, cur1->coef);
107             cur1 = cur1->next;
108         }
109         else if (cur1->degree < cur2->degree) {
110             now->next = oneTerm(cur2->degree, cur2->coef);
111             cur2 = cur2->next;
112         }
113         else {
114             now->next = oneTerm(cur1->degree, cur1->coef + cur2->coef);
115             cur1 = cur1->next;
116             cur2 = cur2->next;
117         }
118         now = now->next;  // move now to its next
119     }
```

```
120
121     // if one polynomial reach its end, add the rest of another to the result
122     if (!cur1) {
123         now->next = cur2; // link now->next to rest of the polynomial
124     }
125     else {
126         now->next = cur1; // link now->next to rest of the polynomial
127     }
128
129     return front;         // return the pointer of the first term of result
130 }
131
132 POLY *sub(POLY *p1, POLY *p2)
133     // compute p1 - p2 and return the result (almost the same as add)
134 {
135     POLY *front;          // pointer to the first term of result polynomial
136     POLY *now;            // pointer to the current term of result polynomial
137     POLY *cur1 = p1;      // pointer to the current term of p1 polynomial
138     POLY *cur2 = p2;      // pointer to the current term of p2 polynomial
139
140     // check conditions and compute the first term of the result polynomial
141     if (cur1->degree > cur2->degree) {
142         front = oneTerm(cur1->degree, cur1->coef);
143         cur1 = cur1->next;
144     }
145     else if(cur1->degree < cur2->degree) {
        else if (cur1->degree < cur2->degree) {
146         front = oneTerm(cur2->degree, -(cur2->coef));
147         cur2 = cur2->next;
148     }
149     else {
150         front = oneTerm(cur1->degree, cur1->coef - cur2->coef);
151         cur1 = cur1->next;
152         cur2 = cur2->next;
153     }
154     now = front;          // point now to front
155
156     // check conditions and compute rest of terms (same as above)
157     while (cur1 && cur2) {
158         if (cur1->degree > cur2->degree) {
159             now->next = oneTerm(cur1->degree, cur1->coef);
```

```
160             cur1 = cur1->next;
161         }
162         else if (cur1->degree < cur2->degree) {
163             now->next = oneTerm(cur2->degree, cur2->coef);
164             cur2 = cur2->next;
165         }
166         else {
167             now->next = oneTerm(cur1->degree, cur1->coef - cur2->coef);
168             cur1 = cur1->next;
169             cur2 = cur2->next;
170         }
171         now = now->next;   // point now to its next
172     }
173 
174     // while one reach its end, do some tansform and add to the result
175     if (!cur1) {
176         while (cur2) {
177             now->next = oneTerm(cur2->degree, -(cur2->coef));
178             cur2 = cur2->next;
179             now = now->next;
180         }
181     }
182     else
183         now->next = cur1; // link now->next to rest of polynomial
184 
185     return front;         // return the first term addres of result polynomial
186 }
187 
188 POLY *mply(POLY *p1, POLY *p2)
189     // use addition to complete multiplication, the below is the description
190     // of algorthim
191     //      steps 1.: multiply a term from p1 with evry term of p2
192     //      steps 2.: add the result above to the result polynomial
193     //      steps 3.: move term from p1 to next term
194     //      steps 4.: continue until reach the end of polynomial
195 {
196     POLY *result = oneTerm(0, 0); // build the initial condition of final result
197     POLY *front;                  // pointer to the first term of tmp result
198     POLY *now;                    // pointer to the current term of tmp result
199     POLY *cur1 = p1;              // pointer to the current term of p1
200     POLY *cur2 = p2;              // pointer tom the current term of p2
```

```
201
202
203     while (cur1) {                // multiply every term in p2 with a term in p1
204         front = oneTerm(cur1->degree + cur2->degree, cur1->coef * cur2->coef);
205         cur2 = cur2->next;
206         now = front;
207         while (cur2) {
208             now->next = oneTerm(cur1->degree + cur2->degree, cur1->coef *\
            No need to use line continuation
209                                 cur2->coef);
210             cur2 = cur2->next;
211             now = now->next;
212         }
213
214         result = add(result, front);  // add tmp result with final result
215         release(front);               // release memory of tmp result
    Memory leaks!
216         cur2 = p2;                    // point cur2 back to first term of p2
217         cur1 = cur1->next;            // point term in cur1 to next term
218     }
219
220     return result;                    // return final result polynomial
221 }
222
223 void print(POLY *p1)
224     // function that print out the polynomial
225 {
226     POLY *cur = p1;                  // point cur to the first term of p1
227
228     if (cur->coef != 1)              // print the first term
229         printf(" %lg x", cur->coef);
230     else
231         printf(" x");
232     if (cur->degree != 1)
233         printf("^%d", cur->degree);
234
235     cur = cur->next;                 // move cur to next term
236     while (cur) {                    // print the rest of terms
237         if (cur->coef != 0) {
238             if (cur->degree > 1) {
239                 if (cur->coef < 0)
```

```
240                     printf(" %lg x^%d", cur->coef, cur->degree);
241                 else if (cur->coef == 1)
242                     printf(" +x^%d", cur->degree);
243                 else
244                     printf(" +%lg x^%d", cur->coef, cur->degree);
245             }
246             else if (cur-> degree == 1) {
            else if (cur->degree == 1) {
247                 if (cur->coef < 0)
248                     printf(" %lg x", cur->coef);
249                 else if (cur->coef == 1)
250                     printf(" +x");
251                 else
252                     printf(" +%lg x", cur->coef);
253             }
254             else {
255                 if (cur->coef < 0)
256                     printf(" %lg", cur->coef);
257                 else
258                     printf(" +%lg", cur->coef);
259             }
260         }
261         cur = cur->next;            // move cur to next term
262     }
263     printf("\n");                   // print next line
264 }
265
266 void release(POLY *p1)
267     // use "free" command to realease memory of un-used polynomial
268 {
269     POLY *cur;                      // a pointer help releasing memory
    Need a blank line here.
270     while (p1) {                    // do until p1 reach its end (NULL)
271         cur = p1->next;             // point cur to p1->next
272         free(p1);                   // release memory in p1
273         p1 = cur;                   // move p1 to cur (p1->next)
274     }
275 }
```