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: 0.00595779 sec
score: 70
o. [Output] Program output is correct, good.
o. [Format] Program format can be improved
o. [Coding] lab12.c spelling errors: polynomail(3), ponits(1), pq(2), suming(1)
o. [add] function can be improved.
o. [sub] function can be improved.
o. [mply] function can be improved.
```

lab12.c

```
1 // EE231002 Lab12. Polynomials
2 // 109061158, 簡佳吟
3 // Date: 2020/12/21
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 typedef struct sPoly {
9
10     int degree;         // the degree of the node
11     double coef;        // the coefficient of the node
12     struct sPoly *next; // pointer to the next node
13
14  }POLY;
   } POLY;
15
16 POLY *oneterm(int degree, double coef);
17     // This function creates a 1-term polynomial of the form coef * X^degree
18     // and returns the new polynomial.
19 POLY *add(POLY *p1, POLY *p2);
20     // This function adds two polynomials pq and p2 to form a new polynomial
21     // and return the new polynomail.
22 POLY *sub(POLY *p1, POLY *p2);
23     // This function subtracts polynomial p2 from p1 to form a new polynomial
24     // and return the new polynomial.
25 POLY *mply(POLY *p1, POLY *p2);
26     // This function multiplies two polynomials p1 and p2 to form a new
27     // polynomial and return the new polynomial.
28 void print(POLY *p1);
29     // This function prints out the polynomial p1 in human readable form.
30     // See the example output given below foe more details.
31 void release(POLY *p1);
32     // This function releases all node of the polynomial p1
33     // and returns them to the heap space.
34 int main(void)
35 {
36
37     POLY *x, *one;                  // x is polynomial x, one is constant 1

   This line has more than 80 characters
```

```
38      POLY *A, *A2, *A3, *A4, *A5;
39      POLY *B, *B2, *B3, *B4, *B5;
40      POLY *C, *C2, *C3, *C4, *C5;
41
42      x = oneterm(1, 1);      // creates polynomials
43      one = oneterm(0, 1);
44      A = add(x, one);
45      A2 = mply(A ,A);
        A2 = mply(A, A);
46      A3 = mply(A2, A);
47      A4 = mply(A3, A);
48      A5 = mply(A4, A);
49
50      printf("A =");          // prompt
51      print(A);
52      printf("A2 =");
53      print(A2);
54
55      B = sub(x, one);        // creates polynomials
56      B2 = mply(B, B);
57      B3 = mply(B2, B);
58      B4 = mply(B3, B);
59      B5 = mply(B4, B);
60
61      C = add(A, B);          // creates polynomials
62      C2 = mply(A2, B2);
63      C3 = mply(A3, B3);
64      C4 = mply(A4, B4);
65      C5 = mply(A5, B5);
66
67      printf("C = ");         // prompt
68      print(C);
69      release(A);             // release
70      release(B);
71      release(C);
72      printf("C2 =");         // prompt
73      print(C2);
74      release(A2);            //release
        release(A2);            // release
75      release(B2);
76      release(C2);
```

```
77      printf("C3 =");         // prompt
78      print(C3);
79      release(A3);            // release
80      release(B3);
81      release(C3);
82      printf("C4 =");         // prompt
83      print(C4);
84      release(A4);            // release
85      release(B4);
86      release(C4);
87      printf("C5 =");         // prompt
88      print(C5);
89      release(A5);            // release
90      release(B5);
91      release(C5);
92
93      release(x);             // release
94      release(one);
95
96      return 0;
97  }
98
99  // This function creates a 1-term polynomial of the form coef * X^degree
100 // and returns the new polynomial.
101 POLY *oneterm(int degree, double coef)
102 {
103     POLY *new_node;
104
105     new_node = (POLY *) malloc(sizeof(POLY));   // creates a space for new_node
106     new_node->degree = degree;                  // assign degree
107     new_node->coef = coef;                      // assign coef
108     new_node->next = NULL;                      // assign next to NULL
109
110     return new_node;                            // done and return
111
112 }
113
114 // This function adds two polynomials pq and p2 to form a new polynomial
115 // and return the new polynomail.
116 POLY *add(POLY *p1, POLY *p2)
117 {
```

```
118     POLY *first;        // point to the new space
119     POLY *s, *t;        // s points to p1, t points to p2
120     POLY *new_node;     // for creating new POLY
121     int maxdegree;      // the maximum degree of two polynomial
122     int coef = 0;       // for suming coef
123     int i;              // index for loop
124
125     first = NULL;
126     maxdegree = (p1->degree >= p2->degree ? p1->degree : p2->degree);
127         // decide the maximum degree of the polynomial to return
128
129     for (i = 0; i <= maxdegree; i++) {              // make new nodes
130         coef = 0;                                   // reset coef
131         for (s = p1; s != NULL; s = s->next) {      // p1's coef of degree
132             if (s->degree == i) coef += s->coef;    // sum the coef
133         }
134         for (t = p2; t != NULL; t = t->next) {      // p2's coef of degree
135             if (t->degree == i) coef += t->coef;    // sum the coef
136         }
137         if (coef != 0 ) {                               // creates new_node
          if (coef != 0) {                                // creates new_node
138             new_node = (POLY *) malloc(sizeof(POLY));   // creates space
139             new_node->degree = i;                       // assign degree
140             new_node->coef = coef;                      // assign coef
141             new_node->next = first;                     // assign next to NULL
142             first = new_node;                           // let first point to
143                                                         // the next new POLY
144         }
145
146     }
147
148     return new_node;                                // done and return
149
150 }
151
152 // This function subtracts polynomial p2 from p1 to form a new polynomial
153 // and return the new polynomial.
154
155 POLY *sub(POLY *p1, POLY *p2)
156 {
157     POLY *first;        // point to the new space
```

```
158     POLY *s, *t;        // s points to p1, t points to p2
159     POLY *new_node;     // for creating POLY
160     int maxdegree;      // the maximum degree of two polynomial
161     int coef = 0;       // for calculating coef
162     int i;              // index for loop
163
164     first = NULL;
165     maxdegree = (p1->degree >= p2->degree ? p1->degree : p2->degree);
166         // decide the maximum degree of the polynomial to return
167
168     for (i = 0; i <= maxdegree; i++) {              // make new nodes
169         coef = 0;                                   // reset coef
170         for (s = p1; s != NULL; s = s->next) {      // p1's coef of degree
171             if (s->degree == i) coef += s->coef;    // calculate the coef
172         }
173         for (t = p2; t != NULL; t = t->next) {      // p2's coef of degree
174             if (t->degree == i) coef -= t->coef;    // calculate the coef
175         }
176         if (coef != 0 ) {                               // make new nodes
          if (coef != 0) {                                // make new nodes
177             new_node = (POLY *) malloc(sizeof(POLY));   // creates space
178             new_node->degree = i;                       // assign degree
179             new_node->coef = coef;                      // assign coef
180             new_node->next = first;                     // assign next to NULL
181             first = new_node;                           // let first point to
182                                                         // the next new POLY
183         }
184
185     }
186     return new_node;                                // done and return
187 }
188
189 // This function multiplies two polynomials p1 and p2 to form a new
190 // polynomial and return the new polynomial.
191 POLY *mply(POLY *p1, POLY *p2)
192 {
193     POLY *first;                // point to the new space
194     POLY *s, *t;                // s ponits p1, t points to p2
195     POLY *new_node;             // for creating new node
196     int maxdegree;              // the maximum degree of the polynomial
197                                 // to return
```

```
198     int coef;                   // calculate coef
199     int i;                      // index for loop
200
201     first = NULL;
202     maxdegree = p1->degree + p2->degree;
203         // decide the maximum degree of the polynomail to return
204     for (i = 0; i <= maxdegree; i++) {              // make new nodes
205         coef = 0;                                   // reset coef
206         for (s = p1; s != NULL; s = s->next) {
207             for (t = p2; t != NULL; t = t->next) {
208                 if (s->degree + t->degree == i) {
209                     coef += (s->coef) *(t->coef);   // calculate coef
                        coef += (s->coef) * (t->coef);   // calculate coef
210                 }
211             }
212         }
213         if (coef != 0) {                                // make new node
214             new_node = (POLY *)malloc(sizeof(POLY));    // create new space
215             new_node->degree = i;                       // assign degree
216             new_node->coef = coef;                      // assign coef
217             new_node->next = first;                     // assign next to NULL
218             first = new_node;                           // let first point to
219                                                         // the new node
220         }
221     }
222     return new_node;                                // done and return
223 }
224
225 // This function prints out the polynomial p1 in human readable form.
226 // See the example output given below foe more details.
227 void print(POLY *p1)
228 {
229     POLY *s;    // point to p1
230
231     for (s = p1; s!= NULL && s->degree >= 1; s = s->next) {
        for (s = p1; s != NULL && s->degree >= 1; s = s->next) {
232         if (s->coef == 1) printf(" x");                 // hide the coef
233         else printf("%g x", s->coef);                   // print its coef
234         if (s->degree == 1) printf(" ");                // hide its degree
235         else printf("^%d ", s->degree);                 // print its degree
236         if ((s->next) != NULL && (s->next)->coef > 0) printf("+");  // print +
```

```
237     }
238     if (s != NULL && s->coef != 0) printf("%g", s->coef);   // print constant
239     printf("\n");
240 }
241
242 // This function releases all node of the polynomial p1
243 // and returns them to the heap space.
244 void release(POLY *p1)
245 {
246     POLY *s, *first;
247
248     first = NULL;
249     s = p1;             // let s point to p1
250     while (s != NULL) {
251         first = s;      // let first point to p1
252         s = s->next;    // let s point to s->next
253         free(first);    // release the node first points to
254     }
255 }
256
```