資料結構報告

楊政愷

August 24, 2024

目錄

1. 解題說明

使用有標頭的串列來時做多項式的各項功能

Homework 3

[***Programming Project***] Develop a C++ class *Polynomial* to represent and manipulate univariate polynomials with integer coefficients (use circular linked lists with header nodes). Each term of the polynomial will be represented as a node. Thus, a node in this system will have three data members as below:

coef	exp	link

Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an available-space list and associated functions as described in Section 4.5. The external (i.e., for input or output) representation of a univariate polynomial will be assumed to be a sequence of integers of the form: $n, c_1, e_1, c_2, e_2, c_3, e_3, \ldots, c_n, e_n$, where e_i represents an exponent and c_i a coefficient; n gives the number of terms in the polynomial. The exponents are in decreasing order—$e_1 > e_2 > \cdots > e_n$.

Write and test the following functions:

(a) *istream*& **operator>>**(*istream*& *is*, *Polynomial*& *x*): Read in an input polynomial and convert it to its circular list representation using a header node.

(b) *ostream*& **operator<<**(*ostream*& *os*, *Polynomial*& *x*): Convert *x* from its linked list representation to its external representation and output it.

(c) *Polynomial*::*Polynomial*(**const** *Polynomial*& *a*) [Copy Constructor]: Initialize the polynomial ***this** to the polynomial *a*.

(d) **const** *Polynomial*& *Polynomial*::**operator=**(**const** *Polynomial*& *a*) **const** [Assignment Operator]: Assign polynomial *a* to ***this**.

(e) *Polynomial*::~*Polynomial*() [Destructor]: Return all nodes of the polynomial ***this** to the available-space list.

(f) *Polynomial* **operator+** (**const** *Polynomial*& *b*) **const** [Addition]: Create and return the polynomial ***this** + *b*.

(g) *Polynomial* **operator**− (**const** *Polynomial*& *b*) **const** [Subtraction] : Create and return the polynomial ***this** − *b*.

(h) *Polynomial* **operator***(**const** *Polynomial*& *b*) **const** [Multiplication]: Create and return the polynomial ***this** * *b*.

(i) **float** *Polynomial*::*Evaluate*(**float** *x*) **const**: Evaluate the polynomial ***this** at *x* and return the result.

2. 演算法設計與實作

資料結構 > HW > code > L_Polynomial.cpp > main()

```cpp
#include <iostream>
#include <cmath>
using namespace std;

struct Node {
    int coef;
    int exp;
    Node* link;

    Node(int c = 0, int e = 0, Node* l = nullptr) : coef(c), exp(e), link(l) {}
};

class Polynomial {
private:
    Node* head;

public:
    Polynomial() {
        head = new Node();
        head->link = head;
    }

    Polynomial(const Polynomial& a) {
        head = new Node();
        head->link = head;
        Node* current = a.head->link;
        Node* last = head;
        while (current != a.head) {
            last->link = new Node(current->coef, current->exp);
            last = last->link;
            current = current->link;
```

Figure2.1.1:L_Polynomial.cpp

資料結構 > HW > code > L_Polynomial.cpp > main()

```
13  class Polynomial {
34      }
32          }
33          last->link = head;
34      }
35
36      ~Polynomial() {
37          Node* current = head->link;
38          while (current != head) {
39              Node* temp = current;
40              current = current->link;
41              delete temp;
42          }
43          delete head;
44      }
45
46      const Polynomial& operator=(const Polynomial& a) {
47          if (this != &a) {
48              this->~Polynomial();
49              head = new Node();
50              head->link = head;
51              Node* current = a.head->link;
52              Node* last = head;
53              while (current != a.head) {
54                  last->link = new Node(current->coef, current->exp);
55                  last = last->link;
56                  current = current->link;
57              }
58              last->link = head;
59          }
60          return *this;
```

Figure2.1.2:L_Polynomial.cpp

資料結構 > HW > code > L_Polynomial.cpp > main()

```
175 };
 46     const Polynomial& operator=(const Polynomial& a) {
 61     }
 62
 63     friend istream& operator>>(istream& is, Polynomial& x) {
 64         int n, c, e;
 65         is >> n;
 66         for (int i = 0; i < n; ++i) {
 67             is >> c >> e;
 68             Node* newNode = new Node(c, e);
 69             Node* current = x.head;
 70             while (current->link != x.head && current->link->exp > e) {
 71                 current = current->link;
 72             }
 73             newNode->link = current->link;
 74             current->link = newNode;
 75         }
 76         return is;
 77     }
 78
 79     friend ostream& operator<<(ostream& os, const Polynomial& x) {
 80         Node* current = x.head->link;
 81         while (current != x.head) {
 82             os << current->coef << "x^" << current->exp;
 83             current = current->link;
 84             if (current != x.head) os << " + ";
 85         }
 86         return os;
 87     }
 88
 89     Polynomial operator+(const Polynomial& b) const {
```

Figure2.1.3:L_Polynomial.cpp

資料結構 > HW > code > L_Polynomial.cpp > main()

```
90          Polynomial result;
91          Node* aPtr = head->link;
92          Node* bPtr = b.head->link;
93          Node* rPtr = result.head;
94
95          while (aPtr != head || bPtr != b.head) {
96              if (aPtr == head || (bPtr != b.head && bPtr->exp > aPtr->exp)) {
97                  rPtr->link = new Node(bPtr->coef, bPtr->exp);
98                  bPtr = bPtr->link;
99              } else if (bPtr == b.head || (aPtr != head && aPtr->exp > bPtr->exp)) {
100                 rPtr->link = new Node(aPtr->coef, aPtr->exp);
101                 aPtr = aPtr->link;
102             } else {
103                 int sumCoef = aPtr->coef + bPtr->coef;
104                 if (sumCoef != 0) {
105                     rPtr->link = new Node(sumCoef, aPtr->exp);
106                 }
107                 aPtr = aPtr->link;
108                 bPtr = bPtr->link;
109             }
110             rPtr = rPtr->link;
111         }
112         rPtr->link = result.head;
113         return result;
114     }
115
116     Polynomial operator-(const Polynomial& b) const {
117         Polynomial result;
118         Node* aPtr = head->link;
119         Node* bPtr = b.head->link;
120         Node* rPtr = result.head;
```

Figure2.1.4:L_Polynomial.cpp

資料結構 > HW > code > L_Polynomial.cpp > main()

```
121
122         while (aPtr != head || bPtr != b.head) {
123             if (aPtr == head || (bPtr != b.head && bPtr->exp > aPtr->exp)) {
124                 rPtr->link = new Node(-bPtr->coef, bPtr->exp);
125                 bPtr = bPtr->link;
126             } else if (bPtr == b.head || (aPtr != head && aPtr->exp > bPtr->exp)) {
127                 rPtr->link = new Node(aPtr->coef, aPtr->exp);
128                 aPtr = aPtr->link;
129             } else {
130                 int diffCoef = aPtr->coef - bPtr->coef;
131                 if (diffCoef != 0) {
132                     rPtr->link = new Node(diffCoef, aPtr->exp);
133                 }
134                 aPtr = aPtr->link;
135                 bPtr = bPtr->link;
136             }
137             rPtr = rPtr->link;
138         }
139         rPtr->link = result.head;
140         return result;
141     }
142
143     Polynomial operator*(const Polynomial& b) const {
144         Polynomial result;
145         Node* aPtr = head->link;
146
147         while (aPtr != head) {
148             Polynomial temp;
149             Node* bPtr = b.head->link;
150             Node* tPtr = temp.head;
```

Figure2.1.5:L_Polynomial.cpp

資料結構 > HW > code > L_Polynomial.cpp > main()

```
152                 while (bPtr != b.head) {
153                     int c = aPtr->coef * bPtr->coef;
154                     int e = aPtr->exp + bPtr->exp;
155                     tPtr->link = new Node(c, e);
156                     tPtr = tPtr->link;
157                     bPtr = bPtr->link;
158                 }
159                 tPtr->link = temp.head;
160                 result = result + temp;
161                 aPtr = aPtr->link;
162             }
163             return result;
164         }
165
166         float Evaluate(float x) const {
167             float result = 0.0;
168             Node* current = head->link;
169             while (current != head) {
170                 result += current->coef * pow(x, current->exp);
171                 current = current->link;
172             }
173             return result;
174         }
175     };
176
177     int main() {
178         Polynomial p1, p2;
179         cout << "First Polynomial(n c1 e1 c2 e2 ... cn en):";
180         cin >> p1;
181         cout << "Second Polynomial(n c1 e1 c2 e2 ... cn en):";
182         cin >> p2;
```

Figure2.1.6:L_Polynomial.cpp

資料結構 > HW > code > L_Polynomial.cpp > main()

```
177  int main() {
182      cin >> p2;
183
184      Polynomial sum = p1 + p2;
185      Polynomial diff = p1 - p2;
186      Polynomial prod = p1 * p2;
187
188      cout << "First Polynomial: " << p1 << endl;
189      cout << "Second Polynomial: " << p2 << endl;
190      cout << "Sum: " << sum << endl;
191      cout << "Diff: " << diff << endl;
192      cout << "Mult: " << prod << endl;
193
194      float x;
195      cout << "x value";
196      cin >> x;
197      cout << "First Polynomial when x = " << x << " value: " << p1.Evaluate(x) << endl;
198
199      return 0;
200  }
201
```

Figure2.1.7:L_Polynomial.cpp

3. 效能分析

F(n) = O(n)

時間複雜度

Operator>>():O(n^2)

Operator<<():O(n)

Polynomial(const Polynomial& a):)O(n)

Operator=():O(n)

~Polynomial():O(n)

Operator+():O(n + m)

Operator-():O(n + m)

Operator*():O)n * m(

Evaluate():))(n)

空間複雜度

Operator>>():O(1)

Operator<<():O(1)

Plynomial():O(n)

Polynomial(const Polynomial& a):)O(n)

Operator=():O(n)

Operator+():O(n + m)

Operator-():O(n + m)

Operator*():O)n * m(

Evaluate():O(1)

4. 測試與過程

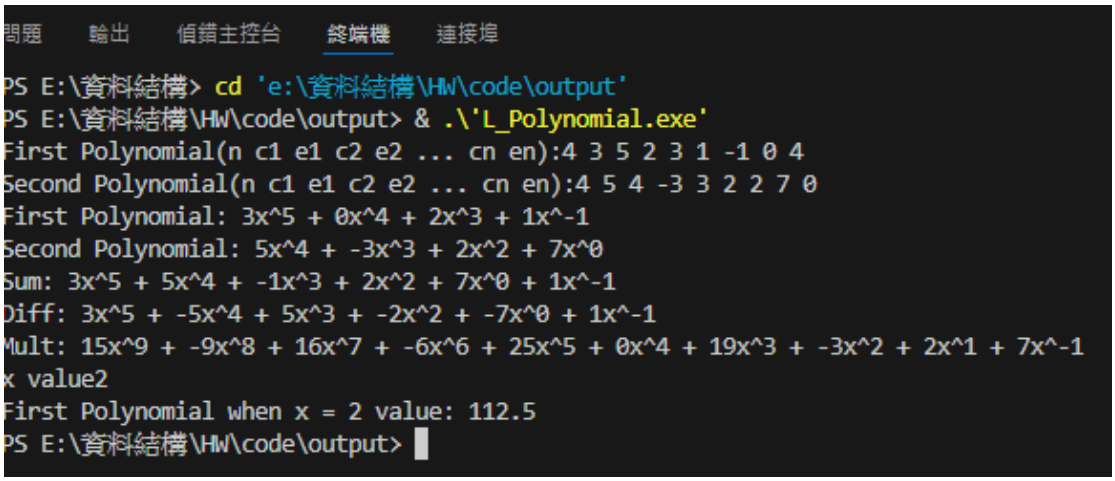

Figure4.1:L_Polynomial.cpp