資料結構作業 3

學號:41043244

姓名:賴明賢

題目一：

[***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.

解題思路:

資料結構設計：

1. **Term 結構體：**
    - 這個結構體表示一個多項式項，包含三個成員：
        - coef：儲存多項式項的係數。
        - exp：儲存多項式項的指數。
        - link：指向下一個 Term 節點的指標（在 LinkNode 中使用）。
2. **LinkNode 類別：**
    - 每個 LinkNode 節點包含一個 Term 對象和一個指向下一個 LinkNode 的指標。這些節點組成一個循環鏈結串列，用於存儲多項式。
3. **LinkList 類別：**
    - 使用 LinkNode 節點的鏈結串列來表示多項式。鏈表包含兩個指標：first 指向頭節點，last 指向最後一個節點。LinkList 支援一些基本操作，如在鏈表末尾插入節點。
4. **Polynomial 類別：**
    - 使用 LinkList 來儲存多項式，並實作各種操作如加法、減法、乘法以及多項式的值計算。

功能實作：

1. **多項式的輸入與輸出 (operator>> 和 operator<<)：**
    - operator>>：從使用者那裡讀取多項式項數和各項的係數及指數，並將其轉換為帶頭節點的循環鏈結串列。
    - operator<<：將鏈結串列形式的多項式轉換回外部表示（例如，輸出格式為 1X^3 + 2X^2 + 3），並輸出。
2. **多項式的加法 (operator+)：**
    - 透過遍歷兩個多項式的鏈結串列，對具有相同指數的項進行加法，並將結果存儲在新的多項式中。最終返回計算後的多項式。
3. **多項式的減法 (operator-)：**
    - 與加法類似，但對於具有相同指數的項，執行的是減法運算。

4. **多項式的乘法（operator*）：**
    - 對每一個多項式中的每一項，分別與另一個多項式的所有項相乘，並將結果累加至新的多項式中。
5. **多項式的值計算（Evaluate）：**
    - 逐項計算每個多項式項在給定 x 值下的結果，並將這些值累加以得到整個多項式的值。
6. **複製建構子與賦值運算子：**
    - 實作了複製建構子（Polynomial::Polynomial(const Polynomial& a)）以及賦值運算子（Polynomial& Polynomial::operator=(const Polynomial& b)）來支援多項式的複製與賦值操作。
7. **析構子（~Polynomial）：**
    - 釋放多項式所佔用的所有記憶體，防止內存洩漏。

程式碼：

```
#include<iostream>
#include<cmath>
#include<Windows.h>
using namespace std;
int counter = 0;
struct Term {
    int coef = -1;
    int exp = -1;
    Term set(int c, int e)
    {
        coef = c;
        exp = e;
        return *this;
    }
};

template<class T>
class LinkList;

template<class T>
class LinkNode {
    friend class LinkList<T>;
private:
    T data;
    LinkNode<T>* link;
public:
    LinkNode(const T& element = T(), LinkNode<T>* next = 0) {
        counter += 2;
        data = element;
        link = next;
    }
};

template<class T>
class LinkList {
private:
    LinkNode<T>* first;
```

```
    LinkNode<T>* last;
public:
    LinkList() {
        counter += 3;
        first = new LinkNode<T>;
        first->link = first;
        last = first;
    }
    ~LinkList() {
        counter++;
        static LinkNode<T>* av;
        if (last) {
            counter += 4;
            LinkNode<T>* first = last->link;
            last->link = av;
            av = first;
            last = 0;
        }
    }

    void InsertBack(const T& e) {
        counter++;
        if (first) {
            counter += 3;
            last->link = new LinkNode<T>(e);
            last = last->link;
            last->link = first;
        }
        else {
            counter++;
            first = last = new LinkNode<T>(e);
        }
    }
    class iterator {
    public:
        iterator(LinkNode<T>* startNode = 0) {
            counter++;
            current = startNode;
        }

        T& operator*() const {
            counter++;
            return current->data;
        }
        T* operator->() const {
            counter++;
            return &current->data;
        }
        iterator& operator ++() {
            counter++;
            current = current->link;
            return *this;
        }
        iterator operator ++(int) {
            counter += 2;
            iterator old = *this;
            current = current->link;
            return *this;
        }
        bool operator!=(const iterator right) const {
            counter++;
            return current != right.current;
        }
        bool operator ==(const iterator right) const {
            counter++;
            return current == right.current;
        }
    private:
        LinkNode<T>* current;
    };
    iterator begin() const {
        counter++;
```

```
110             return iterator(first->link);
111         }
112         iterator end() const {
113             counter++;
114             return iterator(first);
115         }
116     };
117
118     class Polynomial {
119     private:
120         LinkList<Term> poly;
121     public:
122         Polynomial() {};
123         Polynomial operator+(const Polynomial& b) {
124             Term temp;
125             LinkList<Term>::iterator ai = poly.begin(), bi = b.poly.begin();
126             Polynomial c;
127             counter += 3;
128             while (1) {
129                 counter++;
130                 if (ai->exp == bi->exp) {
131                     counter++;
132                     if (ai->exp == -1) {
133                         counter++;
134                         return c;
135                     }
136
137                     int sum = ai->coef + bi->coef;
138                     counter++;
139                     if (sum) {
140                         counter += 3;
141                         c.poly.InsertBack(temp.set(sum, ai->exp));
142                         ai++; bi++;
143                     }
144                 }
145                 else if (ai->exp < bi->exp) {
146                     counter += 2;
147                     c.poly.InsertBack(temp.set(bi->coef, bi->exp));
148                     bi++;
149                 }
150                 else {
151                     counter += 2;
152                     c.poly.InsertBack(temp.set(ai->coef, ai->exp));
153                     ai++;
154                 }
155             }
156         }
157
158         Polynomial operator-(const Polynomial& b) {
159             Term temp;
160             LinkList<Term>::iterator ai = poly.begin(), bi = b.poly.begin();
161             Polynomial c;
162             counter += 3;
163             while (1) {
164                 counter++;
165                 if (ai->exp == bi->exp) {
166                     counter++;
167                     if (ai->exp == -1) {
168                         counter++;
169                         return c;
170                     }
171                     counter++;
172                     int sum = ai->coef - bi->coef;
173                     if (sum) {
174                         counter++;
175                         c.poly.InsertBack(temp.set(sum, ai->exp));
176                     }
177                     counter++;
178                     ai++; bi++;
179                 }
180                 else if (ai->exp < bi->exp) {
181                     counter += 2;
```

```
                c.poly.InsertBack(temp.set((-1) * (bi->coef), bi->exp));
                bi++;
            }
            else {
                counter += 2;
                c.poly.InsertBack(temp.set(ai->coef, ai->exp));
                ai++;
            }
        }
    }

    Polynomial operator*(const Polynomial& b) {
        Term temp;
        LinkList<Term>::iterator ai = poly.begin(), bi = b.poly.begin();
        Polynomial c;
        counter += 3;
        while (1) {
            counter++;
            Polynomial c1;
            if (ai->exp < 0) {
                counter++;
                break;
            }
            while (1) {
                counter += 2;
                c1.poly.InsertBack(temp.set((ai->coef) * (bi->coef), bi->exp + ai->exp));
                bi++;
                if (bi->exp < 0)
                {
                    counter += 2;
                    bi = b.poly.begin();
                    c = c + c1;
                    break;
                }
            }
            counter++;
            ai++;
        }
        return c;
    }
    friend istream& operator>>(istream& in, Polynomial& a);
    friend ostream& operator<<(ostream& ot, Polynomial& a);
    Polynomial(const Polynomial& a) {
        counter++;
        poly = a.poly;
    }
    const Polynomial& operator=(const Polynomial& b) {
        counter++;
        this->poly = b.poly;

        return *this;
    }
    ~Polynomial() { }

    float Evaluate(float x) const {
        counter += 2;
        LinkList<Term>::iterator ai = poly.begin();
        float sum = 0.0;

        while (1) {
            counter++;
            if (ai->exp == -1) {
                counter++;
                break;
            }
            counter += 2;
            sum += (ai->coef) * pow(x, ai->exp);
            ai++;
        }
        return sum;
    }
};
```

```
254
255 istream& operator>>(istream& op_in, Polynomial& a) {
256     counter++;
257     Term temp;
258     int n, c, e;
259     counter++;
260     cout << "輸入項數:" << endl;
261     cin >> n;
262     cout << "輸入係數及次方 (ex:1 3 2 2 4 1)" << endl;
263     while (n) {
264         counter++;
265         op_in >> c >> e;
266         a.poly.InsertBack(temp.set(c, e));
267         n--;
268         counter++;
269     }
270     return op_in;
271     counter++;
272 }
273 ostream& operator<<(ostream& op_out, Polynomial& a) {
274     counter++;
275     LinkList<Term>::iterator ai = a.poly.begin();
276     while (1) {
277         counter++;
278         if (ai->exp == -1) {
279             counter++;
280             return op_out;
281         }
282         if (ai->exp == 0) {
283             counter++;
284             op_out << ai->coef;
285         }
286         else {
287             counter++;
288             op_out << ai->coef << "X^" << ai->exp;
289         }
```

```
299 int main() {
300     LARGE_INTEGER start1, end1, start2, end2, fre, start3, end3, start4, end4;
301     double times1, times2, times3, times4;
302
303     Polynomial Polya, Polyb, Polyc, Polyd, Polye;
304
305     cin >> Polya;
306     cin >> Polyb;
307
308     cout << "a=" << Polya << endl;
309     cout << "b=" << Polyb << endl;
310
311     QueryPerformanceFrequency(&fre);
312
313     QueryPerformanceCounter(&start1);
314     Polyc = Polya + Polyb;
315     QueryPerformanceCounter(&end1);
316
317     times1 = ((double)end1.QuadPart - (double)start1.QuadPart) / fre.QuadPart;
318
319     cout << "a+b=" << Polyc;
320     cout << endl;
321
322     QueryPerformanceCounter(&start2);
323     Polyd = Polya - Polyb;
324     QueryPerformanceCounter(&end2);
325
326     times2 = ((double)end2.QuadPart - (double)start2.QuadPart) / fre.QuadPart;
327
328     cout << "a-b=" << Polyd;
329     cout << endl;
330
331     QueryPerformanceCounter(&start3);
332     Polye = Polya * Polyb;
333     QueryPerformanceCounter(&end3);
334     times3 = ((double)end3.QuadPart - (double)start3.QuadPart) / fre.QuadPart;
```

```
    cout << "b=" << Polyb.Evaluate(x);
    cout << endl;
    cout << "a+b=" << Polyc.Evaluate(x);
    cout << endl;
    cout << "a-b=" << Polyd.Evaluate(x);
    cout << endl;
    cout << "a*b=" << Polye.Evaluate(x);
    cout << endl;
    QueryPerformanceCounter(&end4);

    times4 = ((double)end4.QuadPart - (double)start4.QuadPart) / fre.QuadPart;

    counter += 11;

    cout << "total_steps:" << counter << "\n";

    cout << "total_times：" << times1 + times2 + times3 + times4 << "秒" << endl;


}
```

效能分析:

時間複雜度：

- 加法、減法：O(n)
- 乘法：O(n^2)
- Evaluate：O(n)

空間複雜度：

- 一般操作：O(n)
- 乘法：O(n^2)

測試與過程

```
輸入項數:
5
輸入係數及次方 (ex:1 3 2 2 4 1)
4 4 3 3 2 2 1 1 1 0
輸入項數:
4
輸入係數及次方 (ex:1 3 2 2 4 1)
3 3 2 2 1 1 1 0
a=4X^4+3X^3+2X^2+1X^1+1
b=3X^3+2X^2+1X^1+1
a+b=4X^4+6X^3+4X^2+2X^1+2
a-b=4X^4
a*b=12X^7+17X^6+16X^5+14X^4+10X^3+5X^2+2X^1+1
輸入要代入的X值:
3
a=427
b=103
a+b=530
a-b=324
a*b=43981
total_steps:2049
total_times：0.0026347秒
```