Data Structure Program Assignment #7
(Due: PM:6:00, May 4, 2022)

Polynomial operations by template linked list

Instructor: Jiann-Jone Chen

● Introduction

This homework is designed to demonstrate how to implement a polynomial class based on a linked list structure. In this program, linked list and listnode have been upgraded to Chain and Chainnode to comply with the newest textbook content. However, functions and algorithms are unchanged. During the lecture hours, we demonstrated how to add two polynomials, represented by a template linked list (Chain). You had to implement polynomial multiplication and addition operations based on this template linked list. The linked list design is one of the most important programming skills in C++. You are encouraged to design this program totally by yourself, if possible. In **homework program #3**, you designed algorithms and programs to perform polynomial addition and multiplication operations. These algorithms are the same, and you just need to redesign the linked list data structure according to the lecture 4 notes.

● Steps:

1. The algorithm to perform polynomial addition has been introduced in lecture notes Linked List I-47.
2. You can design the (a) template ListNode class; (b) template List class; and (c) ListIterator class; and (d) polynomial class, respectively, according to the contents shown in lecture notes. For example, ListNode and List (lecture notes I 11, 15), ListIterator(lecture notes I-22~25), polynomial (lecture notes I 45, 47, 49).
3. Design an available list to provide node memory space for other functions.
4. The main program is shown below:

```
#include "ChainNode.h"
#include "Polynomial.h"
#include "av_list.h"
int debug;      // debug = 1 to see debugging information
int N ;   //  N is the number of available nodes initialzed
int main() {
     int cont = 1; LOGO();
     while (cont) {
          Polynomial a, b, c, d;
          ifstream inFile("polynomial.txt", ios::in); //"equiv.in" is th input file
          if (!inFile) { cerr << "Cannot open input file" << endl; return 0; }
          inFile >> a; // input polynomial a data
          inFile >> b; // input polynomial b data

          c = (a + b);
          cout << "a = "<<a<<"\nb = "<<b<<"\nc = a + b =\n"
          << a << " + " << b << " =\n " << c << "\n";

          d = a * b;
          d = c + d;
          cout <<"\n(a+b) + (a*b) =\n"<< c <<'+'<<a<<" * "<<b<<" =\n"<<d<< "\n";
          cont = 0;
          cout << "Press any key to trigger polynomial destructor\n";
          char ch = getchar();
     }  // this right brace trigger polynomial destructor to free all allcated space
     return 0;
};
```

5. The struct Term and Polynomial class are shown below

```
struct Term {  // all members of Term are public by default
     int coef; //coefficient
     int exp;  // exponential lorder
     Term Set(int c, int e) {
          coef = c; exp = e;  return *this; };
     Term Set(ChainNode <Term> temp) {
          Term data = temp.GetData();
          coef = data.coef; exp = data.exp;
          return *this;
     };
};

class Polynomial {
     friend ostream &operator<<(ostream &, Polynomial &);
     friend istream &operator>>(istream &, Polynomial &);
     friend Polynomial operator+(const Polynomial&, const Polynomial&);

public:
     Polynomial(){ label = 'A' + counter;  num = counter++; message(); };
     Polynomial(char c) { label = c;  num = counter++;  message(); };
     Polynomial(const Polynomial&); // copy constructor
     ~Polynomial();
     void operator=(const Polynomial&);
     Polynomial operator*(const Polynomial&) const;
public:
     Chain <Term> poly;
     char label;
private:
     void message();
     static int counter;
     int num;
};
```

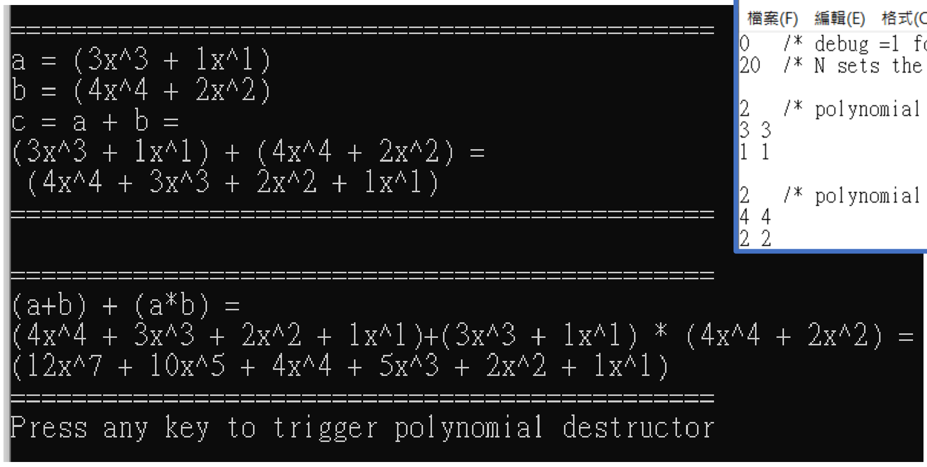
6. A template ChainNode class and a template chain class are provided for your reference. You can overload the new and delete functions to deal with memory allocation operations. (Hint: use static member class to deal with an available list.)

```
  template <class T>
class ChainNode {
     friend class Polynomial;
     friend class Chain <T>;
     friend class ChainIterator <T>; // new added for iterator
     friend istream &operator>>(istream &, Polynomial&);
     template <class T> friend ostream &operator<<(ostream &,  Chain<T>&);
public:
     T GetData() { return data; };
     ChainNode <T>* GetLink() { return link; };
     ChainNode(T element, ChainNode<T>* l = 0) :data(element), link(l) { };
     void* operator new(size_t size) {  // overloading new
          void* p; /* get a node from the av list pointed by p */
          return p;
     }
     void operator delete(void* p) {   // overloading delete
          /* return the node data pointed by p to the av list */
     }
private:
     T data;
     ChainNode<T> *link;
};
```

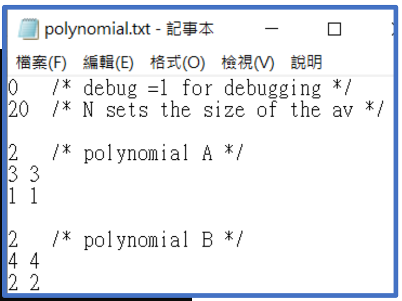
```
// *********Chain Template Class ********************
template<class T>
class Chain {
     template <class T> friend ostream& operator<<(ostream&, Chain<T>&);
     friend class ChainIterator<T>; // New Added for Iterator
     friend class Polynomial;
     friend class ChainNode<T>;
     // friend void *av_init(void*, int);
     friend istream& operator >> (istream&, Polynomial&);  // read in a matrix
public:
     Chain() { current = first = last = NULL; num = 0; }; // constructor
     ~Chain();// destructor to free all allocated memory space, got some trouble
     void Create2(T, T);
     void Insert(ChainNode<T>*, T);
     ChainNode<T>* Pop();
     void Delete(ChainNode<T>*, ChainNode<T>*);
     void InsertBack(const T& e);
     void InsertBack(ChainNode<T>*);
     void InsertInorder(const T& e);
     ChainNode<T>* FirstNode() { return first; };
     void PrintChain();
     void operator=(const Chain<T>& t);
private:
     ChainNode <T>* first, * last, *current;
     int num;
};
```

7. Results:

When the polynomial.txt is set to the one on the right. The correct execution result (set debug=0) looks like the following figure:

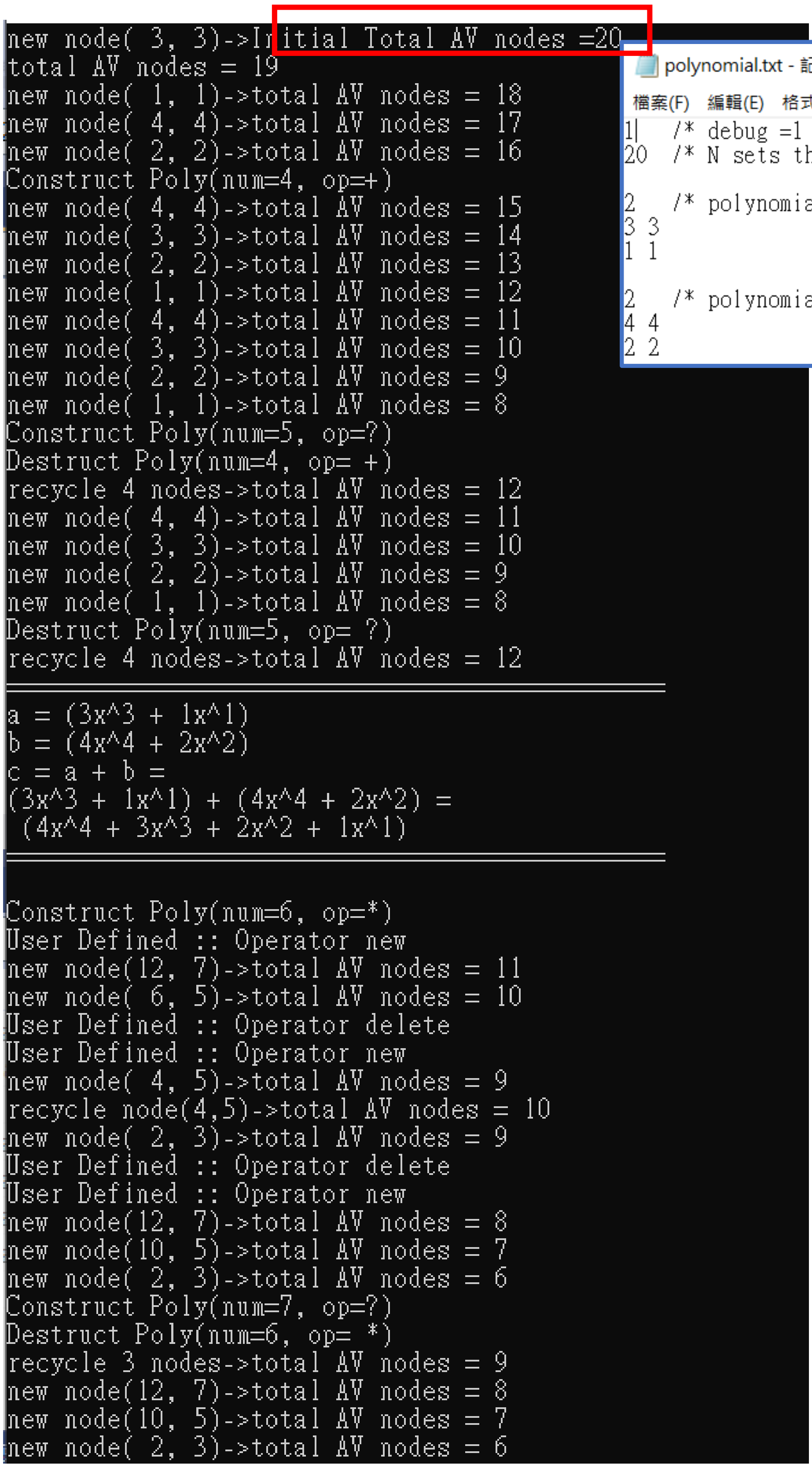


```
==============================================
a = (3x^3 + 1x^1)
b = (4x^4 + 2x^2)
c = a + b =
(3x^3 + 1x^1) + (4x^4 + 2x^2) =
 (4x^4 + 3x^3 + 2x^2 + 1x^1)
==============================================

==============================================
(a+b) + (a*b) =
(4x^4 + 3x^3 + 2x^2 + 1x^1)+(3x^3 + 1x^1) * (4x^4 + 2x^2) =
(12x^7 + 10x^5 + 4x^4 + 5x^3 + 2x^2 + 1x^1)
==============================================
Press any key to trigger polynomial destructor
```

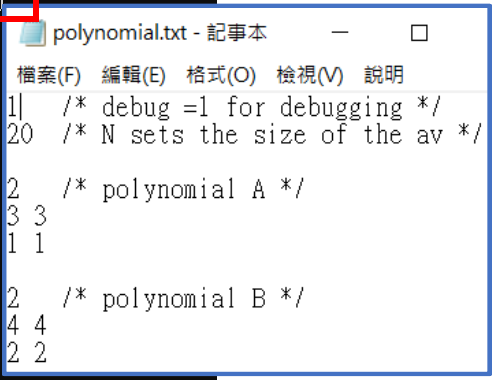


```
0   /* debug =1 for debugging */
20  /* N sets the size of the av */

2   /* polynomial A */
3 3
1 1

2   /* polynomial B */
4 4
2 2
```

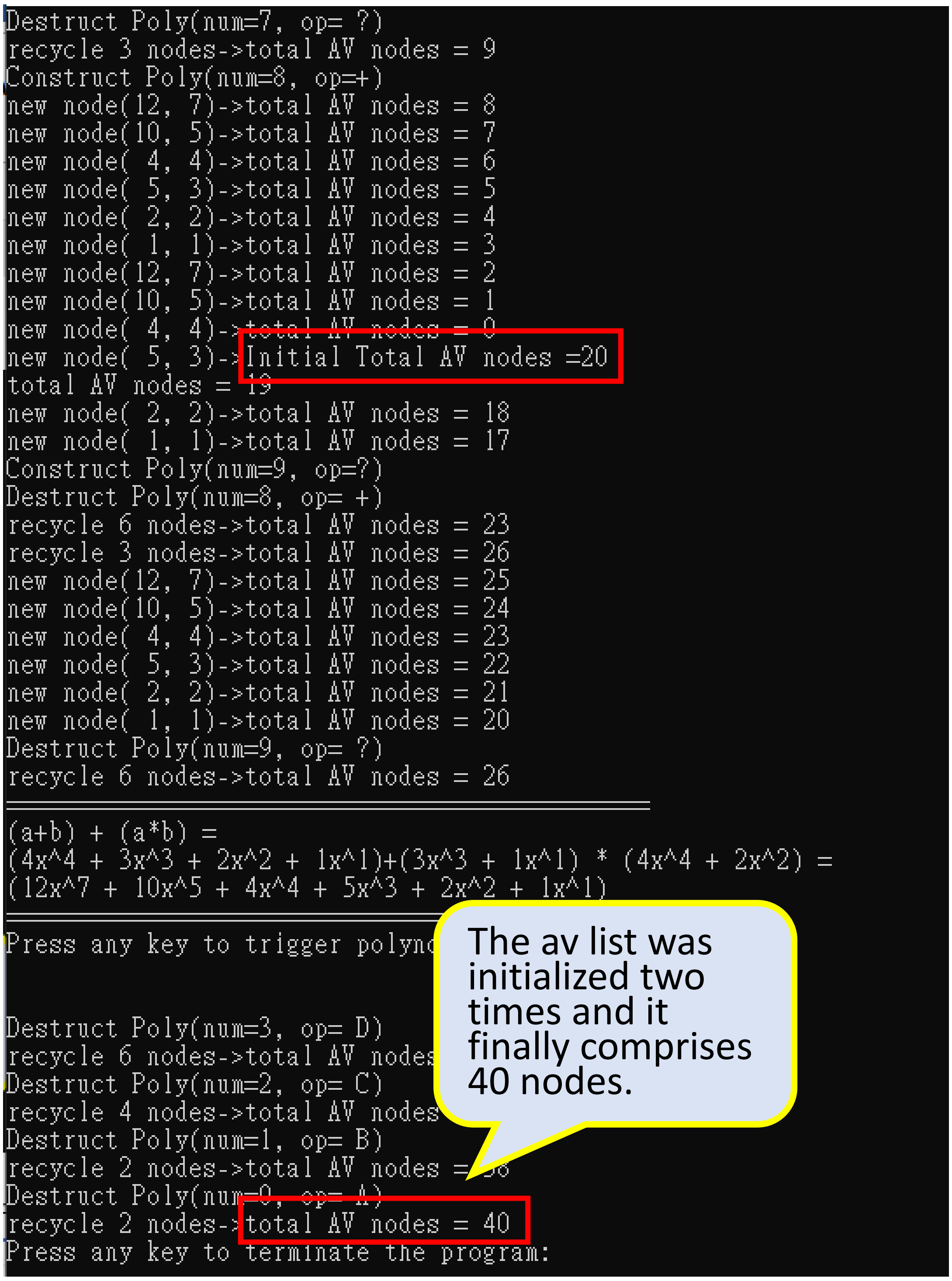
When we set debug=1, the execution result had to show the memory allocation details when your program uses the “new” function to allocate memory for objects.



```
new node( 3, 3)->Initial Total AV nodes =20
total AV nodes = 19
new node( 1, 1)->total AV nodes = 18
new node( 4, 4)->total AV nodes = 17
new node( 2, 2)->total AV nodes = 16
Construct Poly(num=4, op=+)
new node( 4, 4)->total AV nodes = 15
new node( 3, 3)->total AV nodes = 14
new node( 2, 2)->total AV nodes = 13
new node( 1, 1)->total AV nodes = 12
new node( 4, 4)->total AV nodes = 11
new node( 3, 3)->total AV nodes = 10
new node( 2, 2)->total AV nodes = 9
new node( 1, 1)->total AV nodes = 8
Construct Poly(num=5, op=?)
Destruct Poly(num=4, op= +)
recycle 4 nodes->total AV nodes = 12
new node( 4, 4)->total AV nodes = 11
new node( 3, 3)->total AV nodes = 10
new node( 2, 2)->total AV nodes = 9
new node( 1, 1)->total AV nodes = 8
Destruct Poly(num=5, op= ?)
recycle 4 nodes->total AV nodes = 12
==============================================
a = (3x^3 + 1x^1)
b = (4x^4 + 2x^2)
c = a + b =
(3x^3 + 1x^1) + (4x^4 + 2x^2) =
 (4x^4 + 3x^3 + 2x^2 + 1x^1)
==============================================

Construct Poly(num=6, op=*)
User Defined :: Operator new
new node(12, 7)->total AV nodes = 11
new node( 6, 5)->total AV nodes = 10
User Defined :: Operator delete
User Defined :: Operator new
new node( 4, 5)->total AV nodes = 9
recycle node(4,5)->total AV nodes = 10
new node( 2, 3)->total AV nodes = 9
User Defined :: Operator delete
User Defined :: Operator new
new node(12, 7)->total AV nodes = 8
new node(10, 5)->total AV nodes = 7
new node( 2, 3)->total AV nodes = 6
Construct Poly(num=7, op=?)
Destruct Poly(num=6, op= *)
recycle 3 nodes->total AV nodes = 9
new node(12, 7)->total AV nodes = 8
new node(10, 5)->total AV nodes = 7
new node( 2, 3)->total AV nodes = 6
```

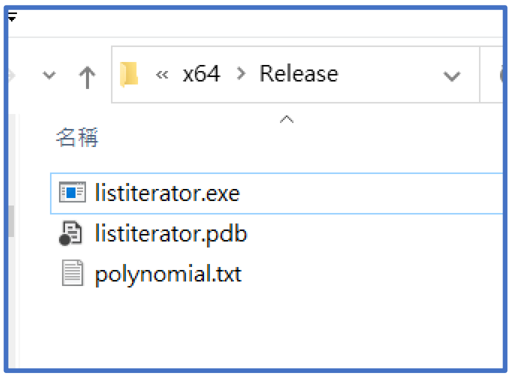


```
1   /* debug =1 for debugging */
20  /* N sets the size of the av */

2   /* polynomial A */
3 3
1 1

2   /* polynomial B */
4 4
2 2
```


Initial Total AV nodes =20
The av list was initialized two times and it finally comprises 40 nodes.

```
Destruct Poly(num=7, op= ?)
recycle 3 nodes->total AV nodes = 9
Construct Poly(num=8, op=+)
new node(12, 7)->total AV nodes = 8
new node(10, 5)->total AV nodes = 7
new node( 4, 4)->total AV nodes = 6
new node( 5, 3)->total AV nodes = 5
new node( 2, 2)->total AV nodes = 4
new node( 1, 1)->total AV nodes = 3
new node(12, 7)->total AV nodes = 2
new node(10, 5)->total AV nodes = 1
new node( 4, 4)->total AV nodes = 0
new node( 5, 3)->
total AV nodes = 19
new node( 2, 2)->total AV nodes = 18
new node( 1, 1)->total AV nodes = 17
Construct Poly(num=9, op=?)
Destruct Poly(num=8, op= +)
recycle 6 nodes->total AV nodes = 23
recycle 3 nodes->total AV nodes = 26
new node(12, 7)->total AV nodes = 25
new node(10, 5)->total AV nodes = 24
new node( 4, 4)->total AV nodes = 23
new node( 5, 3)->total AV nodes = 22
new node( 2, 2)->total AV nodes = 21
new node( 1, 1)->total AV nodes = 20
Destruct Poly(num=9, op= ?)
recycle 6 nodes->total AV nodes = 26
==============================================
(a+b) + (a*b) =
(4x^4 + 3x^3 + 2x^2 + 1x^1)+(3x^3 + 1x^1) * (4x^4 + 2x^2) =
(12x^7 + 10x^5 + 4x^4 + 5x^3 + 2x^2 + 1x^1)
==============================================
Press any key to trigger polyno


Destruct Poly(num=3, op= D)
recycle 6 nodes->total AV nodes
Destruct Poly(num=2, op= C)
recycle 4 nodes->total AV nodes
Destruct Poly(num=1, op= B)
recycle 2 nodes->total AV nodes = 38
Destruct Poly(num=0, op= A)
recycle 2 nodes->total AV nodes = 40
Press any key to terminate the program:
```

9. A demo program,**listiterator.exe**, and a file, **polynomial.txt**, are provided for you to verify your program correctness.


« x64 > Release

名稱

listiterator.exe

listiterator.pdb

polynomial.txt

- **Requirements:**
  1. You will have to submit the complete project such that the TA can recompile your programs to test correctness.
  2. You have to write a short report to describe
     (1) What is all about the program?
     (2) What functions you have designed to provide a calculator program?
     (3) What and how to solve problems during the program design.
     (4) How will you improve this program?
- **Criteria**
  1. (80%) Implement the polynomial class and perform addition and multiplication based on the template linked list class. i.e., your program can output the correct results.
  2. (20%) Implement the memory allocation process based on an available list that can pop and insert nodes to its list to serve as new and delete functions for the template linked list.

- **Hint**
  1. You can write your own function to new and delete node data from the available list. For example, implement the Getnode and the Retnode member functions shown in liked list I-53 to serve new and delete functions. (+10%)
  2. You can also try to design the template listnode and list classes with function overloading to manage the memory allocation for new and delete functions. That is, both new and delete commands are reserved. (+20%)