Bài thực hành số 04. DANH SÁCH LIÊN KẾT ĐƠN

Mục tiêu: thực hành các thao tác trên danh sách liên kết đơn và ứng dụng danh sách liên kết đơn trong thao tác với đa thức, vector thưa và ma trận thưa.

Nội dung thực hành

Bài 1. Danh sách sinh viên

Cho tổ chức dữ liệu danh sách sinh viên như sau:

```
struct Student
{
  string   name;
  int      age;
  double   gpa;
};
struct SLLStudents{
  Student      data;
  SLLStudents  *next;
};
```

Hãy viết các hàm thực hiện các yêu cầu sau và thực thi các hàm:

a) Đọc một danh sách sinh viên từ tệp văn bản, mỗi sinh viên trên một dòng, giữa các trường cách nhau bởi dấu phảy vào một danh sách liên kết đơn lưu danh sách sinh viên.
b) In danh sách sinh viên.
c) Ghi một danh sách sinh viên ra tệp.
d) Thêm một sinh viên vào cuối danh sách
e) Tạo danh sách những sinh viên chứa những sinh viên có gpa >= 8 từ một danh sách cho trước
f) Tìm một sinh viên theo họ tên.
g) Xóa một sinh viên theo họ tên.
h) Thêm một sinh viên vào vị trí k, với k là một số nguyên.
i) Xóa sinh viên tại vị trí k.
j) Sắp xếp danh sách sinh viên theo thứ tự giảm của gpa.
k) Trộn hai danh sách đã sắp theo thứ tự giảm của gpa thành một danh sách đã sắp theo gpa.
l) Chèn danh sách list2 vào danh sách list1 ngay sau sinh viên có tên t.
m) Đảo ngược một danh sách sinh viên.

Hướng dẫn bài 1:

a) Hàm đọc một danh sách sinh viên từ tệp

Đọc từng sinh viên theo dòng rồi thêm vào cuối danh sách.

```
SLLStudent* readFile(const string fileName){
  SLLStudent *head = nullptr, *last = nullptr;
  ifstream inFile(fileName.c_str());
  Student tmp;
  String name, age, gpa;
  while (getline(inFile, name, ',') && getline(inFile, age, ',') &&
                                                    getline(inFile, gpa))
  {
    tmp.name = name; tmp.age = stoint(age); tmp.gpa = stodouble(gpa);
    SLLStudent *p = new SLLStudent;
    p->data = tmp; p->next = nullptr;
    if (head == nullptr) head = p;
    else last->next = p;
    last = p;
  }
  inFile.close();
}
```

```
Nguyen Van A,20,3.5
Tran Thi B,21,3.8
Le Van C,22,2.9
```

b) In danh sách sinh viên.

```
void printListOfStudents(SLLStudents *head)
{



}
```

c) Ghi một danh sách sinh viên ra tệp.

```
void saveFile(SLLStudents *head, const string fileName)
{



}
```

** Sử dụng các hàm:

- Tạo tệp students.txt chứa các sinh viên, mỗi sinh viên trên một dòng.
- Viết hàm main()

```
int main()
{
  SLLStudents *head; //Con trỏ đến phần tử đầu danh sách sinh viên
  //Đọc danh sách sinh viên từ tệp
  head = readFile("students.txt");
  //In danh sách sinh viên vừa đọc
  printListOfStudents(head);
  //Ghi danh sách sinh viên ra tệp students_backup.txt
  saveFile(head, "students_backup.txt");
}
```

d) Thêm một sinh viên vào cuối danh sách
Thuật toán:

```
Cấp phát ô nhớ kiểu SLLStudents cho biến q
Đưa sinh viên cần thêm vào data của q
Cho q liên kết đến rỗng
Nếu danh sách rỗng thì
  Gán q cho head
Ngược lại
  Tìm p là phần tử cuối danh sách
  Cho p liên kết đến q
```

```
void insertLast(SLLStudents* &head, Student s)
{


}
```

e) Tạo danh sách những sinh viên chứa những sinh viên có gpa >= 8 từ một danh sách cho trước
Gợi ý: tạo danh sách mới, duyệt danh sách sinh viên, nếu sinh viên nào có gpa >=8 thì thêm vào cuối danh sách mới.

```
SLLStudents* goodStudents(SLLStudents *head)
{


}
```

f) Tìm một sinh viên theo họ tên.

```
SLLStudents* findByName(SLLStudents *head, string name)
{

}
```

g) Xóa một sinh viên theo họ tên.
Thuật toán:

```
Tìm sinh viên tên x trong danh sách head, gọi p là phần tử tìm được và
p1 là phần tử trước p trong danh sách
Nếu tìm được
  Nếu phần sinh viên cần xóa là phần tử đầu thì xóa phần tử đầu
  Ngược lại thì xóa phần tử sau p1
```

```
void deleteByName(SLLStudents* &head, string name)
{

}
```

h) Thêm một sinh viên vào vị trí k, với k là một số nguyên.
Thuật toán:

```
Tạo q là phần tử của SLLStudent
Đưa s vào dữ liệu của q
Nếu k = 0 thì
  Cho q liên kết đến đầu danh sách
  Gán q cho phần tử đầu
Ngược lại:
  Tìm p là phần tử thứ k-1 trong danh sách
  Cho q liên kết đến sau p
  Cho p liên kết đến q
```

```
void insertStudentByPos(SLLStudents* &head, int k, Student s)
{

}
```

i) Xóa sinh viên tại vị trí k.
Gợi ý: tương tự như thêm sinh viên tại vị trí k.

```
void deleteByPos(SLLStudents* &head, int k)
```

```
{


}
```

j) Sắp xếp danh sách sinh viên theo thứ tự giảm của gpa.

```
void sortByGPA(SLLStudents *head)
{


}
```

k) Trộn hai danh sách đã sắp theo thứ tự giảm của gpa thành một danh sách đã sắp theo gpa.

```
SLLStudents*   mergeListOfStudents(SLLStudents   *head1,   SLLStudents
*head2)
{


}
```

l) Chèn danh sách list2 vào danh sách list1 ngay sau sinh viên có tên name.

```
void  insertListByName(SLLStudents  *list1,  SLLStudents  *list2,  string
name)
{


}
```

m) Đảo ngược một danh sách sinh viên.

Gợi ý thuật toán: duyệt từ đầu đến cuối danh sách, mỗi khi qua một phần tử thì đổi liên kết đến phần tử ngay trước nó để đảo ngược liên kết.

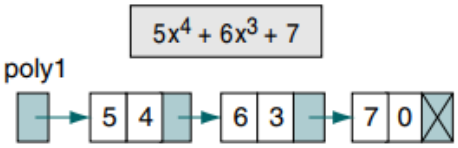
```
SLLStudents* revertList(SLLStudents *head)
{


}
```

Bài 2. Đa thức (Polynomials)

Tổ chức dữ liệu lưu các đa thức $P(x) = a_n x^n + a_{n-1}x^{n-1}+..+a_1x + a_0$ bằng danh sách liên kết đơn như sau:

Mỗi phần tử trong danh sách liên kết đơn là một hệ số a_k và số mũ k với $a_k \neq 0$. Trong đó các phần tử lưu theo thứ tự giảm của số mũ và không lưu các thừa số với hệ số 0 trừ trường hợp đa thức không (P(x) = 0).

Ví dụ:



Hãy khai báo tổ chức dữ liệu và cài đặt các thao tác:

a) Thêm một hạng tử vào đầu đa thức
b) In một đa thức lên màn hình
c) Tính giá trị một đa thức khi cho x một giá trị cụ thể
d) Cộng hai đa thức
e) Tính đạo hàm bậc một của một đa thức.
f) Nhân hai đa thức

Viết chương trình:

- Tạo 2 đa thức $P(x) = 5x^4 + 6x^3 + 7$, $Q(x) = 2x^3 + -7x^2 + 3x$
- In hai đa thức P, Q lên màn hình
- Tính P(2) và in lên màn hình
- Tính R = P + Q, in R lên màn hình.
- Tính D là đa thức đạo hàm bậc một của Q, in D lên màn hình
- Tính M = P × Q, in M lên màn hình

Hướng dẫn bài 2:

Khai báo tổ chức dữ liệu:

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

// Node structure for the linked list
struct Node {
    int coefficient;
    int exponent;
    Node* next;
};
```

a) Hàm thêm một hạng tử vào đầu đa thức: thêm một node vào đầu danh sách.

```
// Function to insert a new node in the linked list
void insert(Node** head, int coef, int expo) {
    Node* newNode = new Node;
    newNode->coefficient = coef;
    newNode->exponent = expo;
    newNode->next = (*head);
    (*head) = newNode;
```

```
}
```

b) Hàm in một đa thức lên màn hình

```
// Function to print the polynomial represented as linked list
void print(Node* head) {
    cout << "The polynomial is: ";
    while (head != NULL) {
        if(head->coefficient < 0){
            cout << "-" << abs(head->coefficient);
        }else{
            cout << "+" << head->coefficient;
        }
        cout << "x^" << head->exponent;
        head = head->next;
    }
    cout << endl;
}
```

c) Tính giá trị đa thức

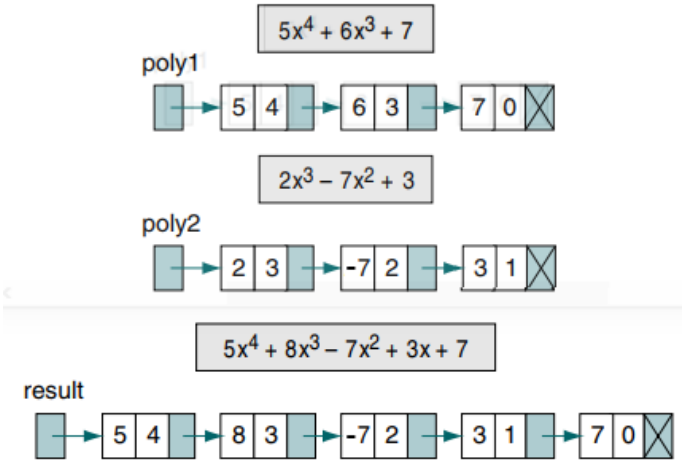
Thuật toán:

```
DL vào: Đa thức p, giá trị x0
DL ra: giá trị đa thức p tại x = x0
Thao tác:
   gt = 0
   Duyệt đa thức p từ đầu đến cuối
      gt = gt + p.coefficient * x0^p.exponent
   Trả về gt
```

Cài đặt:

```
// Function to evaluate the polynomial at a given value
double evaluate(Node* head, double x) {
    double result = 0;
    while (head != NULL) {
        result += head->coefficient * pow(x, head->exponent);
        head = head->next;
    }
    return result;
}
```

d) Cộng hai đa thức



Thuật toán:

DL vào: 2 đa thức p, q
DL ra: đa thức p + q
Thao tác:
  Tạo đa thức kq rỗng
  Duyệt từng phần tử của đa thức p và q khi còn khác rỗng
    Nếu phần tử của đa thức p và q cùng số mũ thì
      Nếu tổng hệ số của hai phần tử tại p và q khác 0 thì
        Thêm vào cuối đa thức kq hạng tử có hệ số là tổng hệ số của hai hạng tử của p và q, số mũ là số mũ của hạng tử đa thức p hoặc q
      Chuyển ra phần tử tiếp theo của đa thức p và q
    Ngược lại nếu số mũ của đa thức nào lớn hơn thì đưa hệ số và số mũ của đa thức đó vào đầu đa thức kq rồi chuyển sang phần tử tiếp theo
  Duyệt từng phần tử còn lại của đa thức p và q thêm vào cuối đa thức kq
Trả về kq

Cài đặt:

Bổ sung hàm `insertNext` thêm một node (hạng tử) vào ngay sau phần tử node.

```
// Function to insert a new node in after the node in the linked list
Node* insertNext(Node* node, int coef, int expo) {
    Node* newNode = new Node;
    newNode->coefficient = coef;
    newNode->exponent = expo;
    newNode->next = NULL;
    if (node) node->next = newNode;
    return newNode;
```

```
}
```

Hàm cộng hai đa thức:

```
// Function to add two polynomials represented as linked lists
Node* add(Node* poly1, Node* poly2) {
    Node *result = NULL, *last = NULL, *temp;

    while (poly1 && poly2) {
        if (poly1->exponent > poly2->exponent) {
         last = insertNext(last, poly1->coefficient,poly1->exponent);
            poly1 = poly1->next;
        }
        else if (poly2->exponent > poly1->exponent) {
           last=insertNext(last,poly2->coefficient,poly2->exponent);
           poly2 = poly2->next;
        }
        else {
            int coef = poly1->coefficient + poly2->coefficient;
            if (coef != 0)
            {
             last = insertNext(last, coef, poly1->exponent);
            }
             poly1 = poly1->next;
           poly2 = poly2->next;
        }
        if (!result)
           result = last;
    }

    while (poly1) {
        last = insertNext(last, poly1->coefficient, poly1->exponent);
        poly1 = poly1->next;
    }

    while (poly2) {
        last = insertNext(last, poly2->coefficient, poly2->exponent);
```

```
        poly2 = poly2->next;
    }

    return result;
}
```

Hàm main thực hiện:

- Tạo 2 đa thức poly1 = $5x^4 + 6x^3 + 7$, poly2 = $2x^3 + -7x^2 + 3x$
- In hai đa thức lên màn hình
- Tính giá trị của đa thức poly1 tại x = 2 và in kết quả lên màn hình
- Tính result = poly1 + poly2, in đa thức result lên màn hình.

```
int main() {
    // Creating the first polynomial
    Node* poly1 = NULL;
    insert(&poly1, 7, 0);
    insert(&poly1, 6, 3);
    insert(&poly1, 5, 4);

    // Creating the second polynomial
    Node* poly2 = NULL;
    insert(&poly2, 3, 1);
    insert(&poly2, -7, 2);
    insert(&poly2, 2, 3);

    // Print two polynomials
    print(poly1);
    print(poly2);

    // Evaluating the first polynomial at x = 2
    double x = 2;
    cout << "The value of the first polynomial at x = " << x << " is: "
<< evaluate(poly1, x) << endl;

    // Adding the two polynomials
    Node* result = add(poly1, poly2);
```

```
    // Displaying the result
    print(result);
    return 0;
}
```

e) Tính đạo hàm của đa thức

Gợi ý:

Tạo đa thức đạo hàm rỗng

Duyệt từng thừa số của đa thức

Nếu thừa số có dạng $a_i x^i$ với i>0 thì đưa phần tử ($i*a_i$, i-1) vào cuối đa thức đạo hàm.

```
Node* derivative(Node* poly) {




}
```

f) Nhân hai đa thức

Gợi ý: để nhân hai đa thức p và q thực hiện

a. Tạo đa thức kq rỗng

b. Duyệt từng thừa số của đa thức p

Duyệt từng thừa số của đa thức q

Cập nhật thừa số (p.coefficient*q.coefficient, p.exponent+q.exponent) vào đa thức kq

Trong đó thao tác cập nhật thực hiện:

Nếu đã có thừa số với số mũ cần thêm trong đa thức thì cộng thêm hệ số vào hệ số đã có của thừa số đó

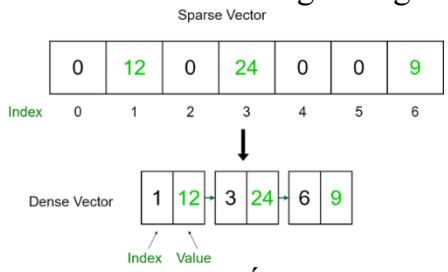
Nếu chưa có thì thêm một thừa số mới vào đa thức kết quả sao cho các thừa số sắp giảm theo số mũ.

```
Node* multiply(Node* poly1, Node* poly2) {
```

```
}
```

Bài 3. Vector thưa (Sparse vector).

Vector thưa là vector mà các thành phần của nó chủ yếu là giá trị 0. Để tiết kiệm bộ nhớ ta dùng danh sách liên kết đơn để lưu vector thưa. Trong đó mỗi phần tử của danh sách liên kết đơn lưu các giá trị khác 0 của vector thưa gồm 2 giá trị (index, value).

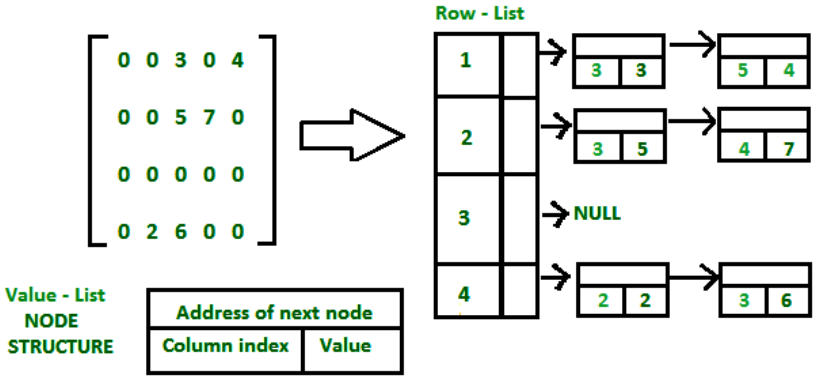


Hãy khai báo tổ chức dữ liệu vector thưa và viết các hàm:

a) Cộng hai véc tơ
b) Nhân một số với một véc tơ
c) Tích vô hướng của hai véc tơ
d) Tính mô đun của một véc tơ
e) Chuyển một vector thành vector thưa
f) Chuyển từ một vector thưa thành vector

*Bài 4. Ma trận thưa 1

Ma trận thưa là ma trận mà các phần tử đa số là số 0. Để lưu ma trận thưa tiết kiệm bộ nhớ ta dùng một danh sách liên kết đơn lưu vector thưa của mỗi dòng.



Hãy khai báo tổ chức dữ liệu cho ma trận thưa và viết các hàm.

a) Cộng, trừ, nhân hai ma trận thưa
b) Chuyển đổi từ ma trận sang ma trận thưa và ngược lại

c) Tích ma trận thưa và vector thưa
