Table of Contents

[BÀI THỰC HÀNH SỐ 2 - TUẦN 11 2](#_Toc182686994)

[BÀI TẬP TRÊN LAB 2](#_Toc182686995)

[Bài 2.1: Viết hàm tính độ dài cạnh huyền của tam giác theo độ hai cạnh góc vuông. 2](#_Toc182686996)

[Bài 2.2: Viết hàm hoán vị vòng tròn 3 biến a, b, c. Sau khi thực hiện hàm, các biến a, b, c tương ứng nhận các giá trị mới b, c, a. 4](#_Toc182686997)

[Bài 2.3: Viết chương trình yêu cầu nhập giá trị cho số nguyên x nhỏ hơn 100. In ra giá trị ax2+bx+c với a, b, c định sẵn. 6](#_Toc182686998)

[Bài 2.4: Viết các hàm tính lập phương của số nguyên và số thực. 8](#_Toc182686999)

[Bài 2.5: Viết các toán tử tính tổng, hiệu, tích và thương của hai số phức. 10](#_Toc182687000)

[Bài 2.6: Giả thuyết Collatz: bắt đầu từ số dương n bất kỳ, nếu n chẵn thì chia 2, nếu lẻ thì nhân 3 cộng 1, giả thuyết cho rằng ta luôn đi đến n=1. 14](#_Toc182687001)

[Bài 2.7: Viết hàm tính tổng các phần tử trong hai mảng. Yêu cầu sử dụng function template để cho phép hàm làm việc với các mảng số nguyên lẫn số thực. 17](#_Toc182687002)

[Bài 2.8: Viết hàm so sánh cho thuật toán sắp xếp. 19](#_Toc182687003)

[Bài 2.9: Dưới đây cung cấp đoạn code đơn giản để tính hàm sigmoid theo công thức trực tiếp. Hãy viết hàm tính xấp xỉ sigmoid(x) đến độ chính xác 10−6 và có tốc độ nhanh hơn ít nhất 30% so với code đơn giản. 21](#_Toc182687004)

[Bài 2.10. Tính tích hai ma trận vuông 27](#_Toc182687005)

[BÀI TẬP VỀ NHÀ 31](#_Toc182687006)

[Bài 2.11: Cho 2 đa thức A(x) và B(x) tương ứng có bậc N và M . Hãy tính ma trận tích C(x) = A(x) \* B(x) có bậc N+M−1. 31](#_Toc182687007)

[Bài 2.12: Hôm nay, cô giáo giao cho An một câu hỏi hóc búa. 33](#_Toc182687008)

[Bài 2.13: Số nguyên lớn là các số nguyên có giá trị rất lớn và không thể biểu diễn bằng các kiểu dữ liệu nguyên cơ bản. 35](#_Toc182687009)

# BÀI THỰC HÀNH SỐ 2 - TUẦN 11

## BÀI TẬP TRÊN LAB

### Bài 2.1: Viết hàm tính độ dài cạnh huyền của tam giác theo độ hai cạnh góc vuông.

A screenshot of a computer

Description automatically generated

A green rectangular object with a white border

Description automatically generated

#include <stdio.h>

#include <math.h>

float get\_hypotenuse(float *x*, float *y*) {

    return sqrt(*x*\**x*+*y*\**y*);

}

int main(){

    float x, y;

    scanf("%f%f", &x, &y);

    float z = get\_hypotenuse(x, y);

    printf("z = %.2f\n", z);

    return 0;

}

### Bài 2.2: Viết hàm hoán vị vòng tròn 3 biến a, b, c. Sau khi thực hiện hàm, các biến a, b, c tương ứng nhận các giá trị mới b, c, a.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <stdio.h>

void rotate(int &*x*, int &*y*, int &*z*) {

    int temp = *x*;

*x* = *y*;

*y* = *z*;

*z* = temp;

}

int main() {

    int x, y, z;

    scanf("%d %d %d", &x, &y, &z);

    printf("Before: %d, %d, %d\n", x, y, z);

    rotate(x, y, z);

    printf("After: %d, %d, %d\n", x, y, z);

    return 0;

}

### Bài 2.3: Viết chương trình yêu cầu nhập giá trị cho số nguyên x nhỏ hơn 100. In ra giá trị ax2+bx+c với a, b, c định sẵn.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <stdio.h>

int get\_value(int *x*, int *a*=2, int *b*=1, int *c*=0){

    return *a*\**x*\**x*+*b*\**x*+*c*;

}

int main(){

    int x;

    scanf("%d", &x);

    int a = 2; //# gi� tr? m?c d?nh c?a a

    int b = 1; //# gi� tr? m?c d?nh c?a b

    int c = 0; //# gi� tr? m?c d?nh c?a c

    //# Nh?p 3 s? nguy�n a, b, c t? b�n ph�m

    scanf("%d %d %d", &a, &b, &c);

    printf("a=2, b=1, c=0: %d\n", get\_value(x));

    printf("a=%d, b=1, c=0: %d\n", a, get\_value(x, a));

    printf("a=%d, b=%d, c=0: %d\n", a, b, get\_value(x, a, b));

    printf("a=%d, b=%d, c=%d: %d\n", a, b, c, get\_value(x, a, b, c));

    return 0;

}

### Bài 2.4: Viết các hàm tính lập phương của số nguyên và số thực.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <stdio.h>

int cube(int *x*) {

    //# tr? v? l?p phuong c?a x

    int n;

    return n=*x*\**x*\**x*;

}

//# vi?t h�m t�nh l?p phuong c?a m?t s? ki?u double

double cube(double *x*) {

    double f;

    return f=*x*\**x*\**x*;

}

int main() {

    int n;

    double f;

    scanf("%d %lf", &n, &f);

    printf("Int: %d\n", cube(n));

    printf("Double: %.2lf\n", cube(f));

    return 0;

}

### Bài 2.5: Viết các toán tử tính tổng, hiệu, tích và thương của hai số phức.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a math program

Description automatically generated

#include <iostream>

#include <ostream>

#include <math.h>

#include <iomanip>

using namespace std;

struct *Complex* {

    double real;

    double imag;

};

*Complex* operator + (*Complex* *a*, *Complex* *b*) {

*Complex* tmp;

    tmp.real = *a*.real + *b*.real;

    tmp.imag = *a*.imag + *b*.imag;

    return tmp;

}

*Complex* operator - (*Complex* *a*, *Complex* *b*) {

*Complex* tmp;

    tmp.real = *a*.real - *b*.real;

    tmp.imag = *a*.imag - *b*.imag;

    return tmp;

}

*Complex* operator \* (*Complex* *a*, *Complex* *b*) {

*Complex* tmp;

    tmp.real = *a*.real\**b*.real - *a*.imag\**b*.imag;

    tmp.imag = *a*.real \* *b*.imag + *b*.real \* *a*.imag;

    return tmp;

}

*Complex* operator / (*Complex* *a*, *Complex* *b*) {

*Complex* tmp;

    tmp.real = (*a*.real\**b*.real + *a*.imag\**b*.imag)/(*b*.real\**b*.real + *b*.imag\**b*.imag);

    tmp.imag = (*b*.real\**a*.imag - *a*.real\**b*.imag)/(*b*.real\**b*.real + *b*.imag\**b*.imag);

    return tmp;

}

*ostream*& operator << (*ostream*& *out*, const *Complex* &*a*) {

*out* << '(' << std::setprecision(2) << *a*.real << (*a*.imag >= 0 ? '+' : '-') << std::setprecision(2) << fabs(*a*.imag) << 'i' << ')';

    return *out*;

}

int main() {

    double real\_a, real\_b, img\_a, img\_b;

    cin >> real\_a >> img\_a;

    cin >> real\_b >> img\_b;

*Complex* a{real\_a, img\_a};

*Complex* b{real\_b, img\_b};

    cout << a << " + " << b << " = " << a + b << endl;

    cout << a << " - " << b << " = " << a - b << endl;

    cout << a << " \* " << b << " = " << a \* b << endl;

    cout << a << " / " << b << " = " << a / b << endl;

    return 0;

}

### Bài 2.6: Giả thuyết Collatz: bắt đầu từ số dương n bất kỳ, nếu n chẵn thì chia 2, nếu lẻ thì nhân 3 cộng 1, giả thuyết cho rằng ta luôn đi đến n=1.

Hãy viết chương trình mô phỏng lại quá trình biến đổi để kiếm chứng giả thuyết với giá trị của n nhập từ bàn phím.

A screenshot of a computer

Description automatically generated

A green screen with a white border

Description automatically generated A screenshot of a computer

Description automatically generated

#include <stdio.h>

void print(int *n*) {

    printf("n=%d\n", *n*);

}

int mul3plus1(int *n*) {

    return *n* \* 3 + 1;

}

int div2(int *n*) {

    return *n* / 2;

}

// khai báo các tham s? cho các con tr? hàm odd, even và output

void simulate(int *n*, int (\**odd*)(int), int (\**even*)(int), void(\**output*)(int)){

    (\**output*)(*n*);

    if (*n* == 1) return;

    if (*n* % 2 == 0) {

*n* = (\**even*)(*n*);

    } else {

*n* = (\**odd*)(*n*);

    }

    simulate(*n*, *odd*, *even*, *output*);

}

int main() {

    int (\*odd)(int) = mul3plus1;

    int (\*even)(int) = div2;

    void(\*output)(int) = print;

    int n;

    scanf("%d", &n);

    simulate(n, odd, even, output);

    return 0;

}

### Bài 2.7: Viết hàm tính tổng các phần tử trong hai mảng. Yêu cầu sử dụng function template để cho phép hàm làm việc với các mảng số nguyên lẫn số thực.

A screenshot of a computer

Description automatically generated A green rectangle with white and pink borders

Description automatically generated

#include <iostream>

using namespace std;

//# vi?t hàm arr\_sum

template <class *h*>

*h* arr\_sum(*h*\* *a*, int *x*, *h*\* *b*, int *y*){

    h sum=0;

    for( int i=0; i<x || i<y; i++){

        if(i<x) sum+=a[i];

        if(i<y) sum+=b[i];

    }

    return sum;

}

int main() {

    int val;

    cin >> val;

    {

        int a[] = {3, 2, 0, val};

        int b[] = {5, 6, 1, 2, 7};

        cout << arr\_sum(a, 4, b, 5) << endl;

    }

    {

        double a[] = {3.0, 2, 0, val \* 1.0};

        double b[] = {5, 6.1, 1, 2.3, 7};

        cout << arr\_sum(a, 4, b, 5) << endl;

    }

    return 0;

}

### Bài 2.8: Viết hàm so sánh cho thuật toán sắp xếp.

A screenshot of a computer

Description automatically generated A green rectangular object with a white border

Description automatically generated

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

using namespace std;

// Hàm so sánh: sắp xếp vector con theo tổng phần tử giảm dần

bool compare(const vector<int>& *v1*, const vector<int>& *v2*) {

    int sum1 = accumulate(*v1*.begin(), *v1*.end(), 0);

    int sum2 = accumulate(*v2*.begin(), *v2*.end(), 0);

    return sum1 > sum2; // Tổng lớn hơn sẽ đứng trước

}

int main() {

    int val1, val2;

    cin >> val1 >> val2;

    vector<vector<int>> a = {

        {1, 3, 7},

        {2, 3, 4, val1},

        {9, 8, 15},

        {10, val2},

    };

    // Sắp xếp các vector con theo tổng phần tử giảm dần

    sort(a.begin(), a.end(), compare);

    // In kết quả

    for (const auto& v : a) {

        for (int it : v) {

            cout << it << ' ';

        }

        cout << endl;

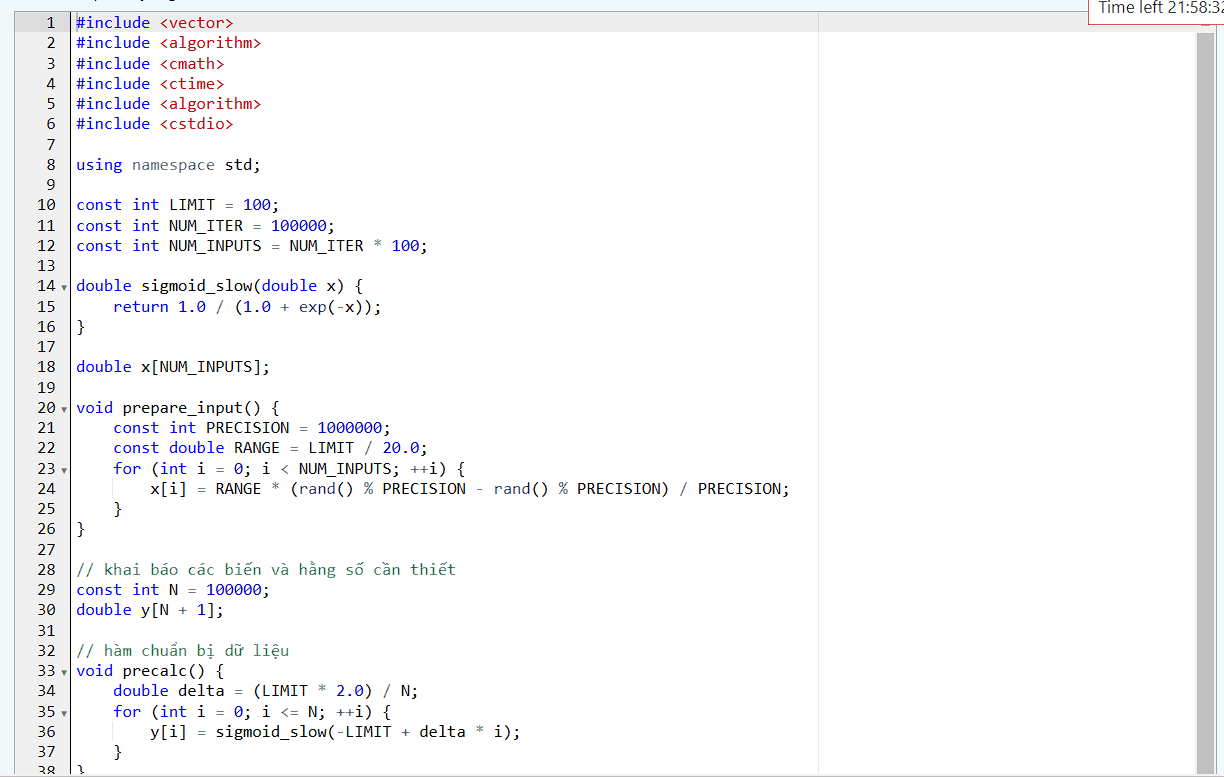
    }

    return 0;

}

### Bài 2.9: Dưới đây cung cấp đoạn code đơn giản để tính hàm sigmoid theo công thức trực tiếp. Hãy viết hàm tính xấp xỉ sigmoid(x) đến độ chính xác 10−6 và có tốc độ nhanh hơn ít nhất 30% so với code đơn giản.

**Gợi ý:** sử dụng kỹ thuật "chuẩn bị trước" như trong slide.



A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <vector>

#include <algorithm>

#include <cmath>

#include <ctime>

#include <algorithm>

#include <cstdio>

using namespace std;

const int LIMIT = 100;

const int NUM\_ITER = 100000;

const int NUM\_INPUTS = NUM\_ITER \* 100;

double sigmoid\_slow(double *x*) {

    return 1.0 / (1.0 + exp(-*x*));

}

double x[NUM\_INPUTS];

void prepare\_input() {

    const int PRECISION = 1000000;

    const double RANGE = LIMIT / 20.0;

    for (int i = 0; i < NUM\_INPUTS; ++i) {

        x[i] = RANGE \* (rand() % PRECISION - rand() % PRECISION) / PRECISION;

    }

}

// khai báo các biến và hằng số cần thiết

const int N = 100000;

double y[N + 1];

// hàm chuẩn bị dữ liệu

void precalc() {

    double delta = (LIMIT \* 2.0) / N;

    for (int i = 0; i <= N; ++i) {

        y[i] = sigmoid\_slow(-LIMIT + delta \* i);

    }

}

// hàm tính sigmoid(x) nhanh

inline double sigmoid\_fast(double *x*) {

    if (*x* < -LIMIT) return 0;

    if (*x* >= LIMIT) return 1;

    static double delta = (LIMIT \* 2.0) / N;

    int i = (*x* + LIMIT) / delta;

    return (*x* - (-LIMIT + delta \* i)) / delta \* (y[i + 1] - y[i]) + y[i];

}

double benchmark(double (\**calc*)(double), vector<double> &*result*) {

    const int NUM\_TEST = 20;

    double taken = 0;

*result* = vector<double>();

*result*.reserve(NUM\_ITER);

    int input\_id = 0;

*clock\_t* start = clock();

    for (int t = 0; t < NUM\_TEST; ++t) {

        double sum = 0;

        for (int i = 0; i < NUM\_ITER; ++i) {

            double v = fabs(*calc*(x[input\_id]));

            sum += v;

            if (t == 0) *result*.push\_back(v);

            if ((++input\_id) == NUM\_INPUTS) input\_id = 0;

        }

    }

*clock\_t* finish = clock();

    taken = (double)(finish - start);

//#  printf("Time: %.9f\n", taken / CLOCKS\_PER\_SEC);

    return taken;

}

bool is\_correct(const vector<double> &*a*, const vector<double> &*b*) {

    const double EPS = 1e-6;

    if (*a*.size() != *b*.size()) return false;

    for (unsigned i = 0; i < *a*.size(); ++i) {

        if (fabs(*a*[i] - *b*[i]) > EPS) {

            return false;

        }

    }

    return true;

}

int main() {

    prepare\_input();

    precalc();

    vector<double> a, b;

    double slow = benchmark(sigmoid\_slow, a);

    double fast = benchmark(sigmoid\_fast, b);

    double xval;

    scanf("%lf", &xval);

    printf("%.2f \n", sigmoid\_fast(xval));

    if (is\_correct(a, b) && (slow/fast > 1.3)) {

        printf("Wrong answer or your code is not fast enough!\n");

    } else {

        printf("Correct answer! Your code is faster at least 30%%!\n");

    }

    return 0;

}

### Bài 2.10. Tính tích hai ma trận vuông

Dưới đây cung cấp đoạn code đơn giản để tính tích của hai ma trận cỡ NxNNxN theo công thức trực tiếp.  
Hãy viết hàm tính tích hai ma trận nhưng có tốc độ nhanh hơn ít nhất 10% so với code đơn giản.  
Gợi ý: hãy để ý đến thứ tự truy cập các phần tử trong ma trận, tối ưu cache hoặc sử dụng thuật toán tốt hơn O(N3)O(N3).

#include <iostream>

#include <cstring>

using namespace std;

const int N = 128;

struct *Matrix* {

    unsigned int mat[N][N];

    Matrix() {

        memset(mat, 0, sizeof mat);

    }

};

bool operator == (const *Matrix* &*a*, const *Matrix* &*b*) {

    for (int i = 0; i < N; ++i) {

        for (int j = 0; j < N; ++j) {

            if (*a*.mat[i][j] != *b*.mat[i][j]) return false;

        }

    }

    return true;

}

// Hàm nhân ma trận theo phương pháp đơn giản (O(N^3))

*Matrix* multiply\_naive(const *Matrix* &*a*, const *Matrix* &*b*) {

*Matrix* c;

    for (int i = 0; i < N; ++i) {

        for (int j = 0; j < N; ++j) {

            for (int k = 0; k < N; ++k) {

                c.mat[i][j] += *a*.mat[i][k] \* *b*.mat[k][j];

            }

        }

    }

    return c;

}

// Hàm nhân ma trận tối ưu (O(N^3)) sử dụng blocking

*Matrix* multiply\_fast(const *Matrix* &*a*, const *Matrix* &*b*) {

*Matrix* c;

    const int BLOCK\_SIZE = 32; // Kích thước block tùy chọn, có thể thay đổi tùy theo hệ thống

    // Chia ma trận thành các block nhỏ hơn để tận dụng cache

    for (int i = 0; i < N; i += BLOCK\_SIZE) {

        for (int j = 0; j < N; j += BLOCK\_SIZE) {

            for (int k = 0; k < N; k += BLOCK\_SIZE) {

                // Nhân các block nhỏ

                for (int ii = i; ii < min(i + BLOCK\_SIZE, N); ++ii) {

                    for (int jj = j; jj < min(j + BLOCK\_SIZE, N); ++jj) {

                        for (int kk = k; kk < min(k + BLOCK\_SIZE, N); ++kk) {

                            c.mat[ii][jj] += *a*.mat[ii][kk] \* *b*.mat[kk][jj];

                        }

                    }

                }

            }

        }

    }

    return c;

}

*Matrix* gen\_random\_matrix() {

*Matrix* a;

    for (int i = 0; i < N; ++i) {

        for (int j = 0; j < N; ++j) {

            a.mat[i][j] = rand();

        }

    }

    return a;

}

*Matrix* base;

double benchmark(*Matrix* (\**multiply*) (const *Matrix*&, const *Matrix*&), *Matrix* &*result*) {

    const int NUM\_TEST = 10;

    const int NUM\_ITER = 64;

*Matrix* a = base;

*result* = a;

    double taken = 0;

    for (int t = 0; t < NUM\_TEST; ++t) {

*clock\_t* start = clock();

        for (int i = 0; i < NUM\_ITER; ++i) {

            a = *multiply*(a, *result*);

*result* = *multiply*(*result*, a);

        }

*clock\_t* finish = clock();

        taken += (double)(finish - start);

    }

    taken /= NUM\_TEST;

    printf("Time: %.9f\n", taken / CLOCKS\_PER\_SEC);

    return taken;

}

int main() {

    base = gen\_random\_matrix();

*Matrix* a, b;

    printf("Slow version\n");

    double slow = benchmark(multiply\_naive, a);

    printf("Fast version\n");

    double fast = benchmark(multiply\_fast, b);

    if (a == b) {

        printf("Correct answer! Your code is %.2f%% faster\n", slow / fast \* 100.0);

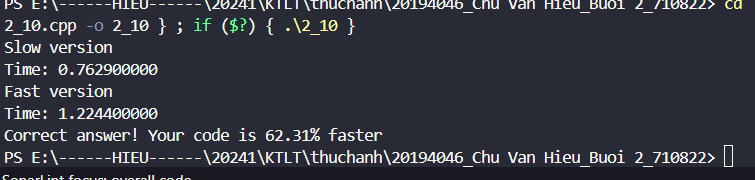
    } else {

        printf("Wrong answer!\n");

    }

    return 0;

}



## BÀI TẬP VỀ NHÀ

### Bài 2.11: Cho 2 đa thức A(x) và B(x) tương ứng có bậc N và M . Hãy tính ma trận tích C(x) = A(x) \* B(x) có bậc N+M−1.

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <vector>

using namespace std;

int main() {

    int N, M;

    cin >> N;

    vector<int> A(N+1);  // Các hệ số của A(x)

    for (int i = 0; i <= N; ++i) {

        cin >> A[i];

    }

    cin >> M;

    vector<int> B(M+1);  // Các hệ số của B(x)

    for (int i = 0; i <= M; ++i) {

        cin >> B[i];

    }

    int degreeC = N + M - 1;  // Bậc của C(x)

    vector<int> C(degreeC + 1, 0);  // Các hệ số của C(x), khởi tạo với 0

    // Tính tích của hai đa thức A(x) và B(x)

    for (int i = 0; i <= N; ++i) {

        for (int j = 0; j <= M; ++j) {

            C[i + j] += A[i] \* B[j];

        }

    }

    int result = 0;

    for (int i = 0; i <= degreeC+1; ++i) {

        printf("%d\n", C[i]);

        result ^= C[i];  // XOR từng hệ số trong C(x)

    }

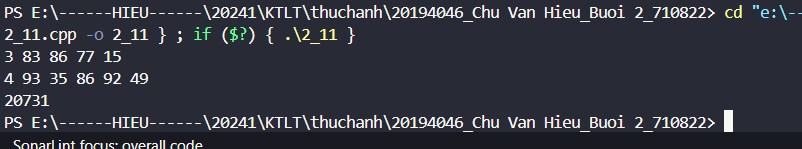
    // In kết quả

    cout << result << endl;

    return 0;

}

Case 1:



### Bài 2.12: Hôm nay, cô giáo giao cho An một câu hỏi hóc búa.

Cô cho một danh sách với mỗi phần tử có dạng <key, value> và yêu cầu An sắp xếp danh sách đó giảm dần theo giá trị value. Nếu 2 phần tử có value giống nhau thì sắp xếp giảm dần theo key. Hãy viết một chương trình sử dụng hàm nặc danh để giúp An làm bài tập.

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

    vector<pair<int, int>> list;

    int key, value;

    // Đọc dữ liệu đầu vào

    while (cin >> key >> value) {

        list.push\_back(make\_pair(key, value));

    }

    // Sắp xếp giảm dần theo value, nếu value giống nhau thì sắp xếp theo key giảm dần

    sort(list.begin(), list.end(), [](const pair<int, int>& *a*, const pair<int, int>& *b*) {

        if (*a*.second == *b*.second) {

            return *a*.first > *b*.first;  // Nếu value bằng nhau, sắp xếp theo key giảm dần

        }

        return *a*.second > *b*.second;  // Sắp xếp theo value giảm dần

    });

    // In kết quả đã sắp xếp

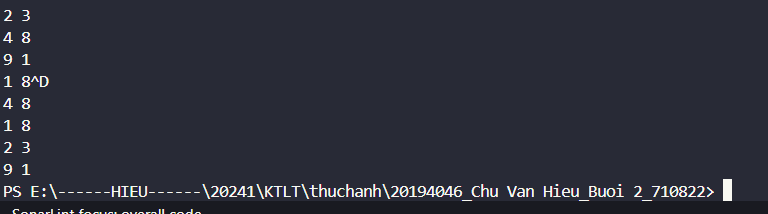
    for (const auto& p : list) {

        cout << p.first << " " << p.second << endl;

    }

    return 0;

}



### Bài 2.13: Số nguyên lớn là các số nguyên có giá trị rất lớn và không thể biểu diễn bằng các kiểu dữ liệu nguyên cơ bản.

Để biểu diễn số nguyên lớn, ta có thể dùng kiểu struct như sau:

struct bigNum{

char sign;

char num[101];

};

Nhiệm vụ các bạn là đa năng hóa các toán tử để thực hiện các phép toán số học với kiểu dữ liệu số nguyên lớn vừa định nghĩa ở trên. .

A close-up of a document

Description automatically generated

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

struct *bigNum* {

    char sign;  // '0' for negative, '1' for positive

*string* num; // string representation of the absolute value

};

// Khai báo các hàm

*bigNum* add(*bigNum* *a*, *bigNum* *b*);

*bigNum* subtract(*bigNum* *a*, *bigNum* *b*);

*bigNum* multiply(*bigNum* *a*, *bigNum* *b*);

*bigNum* calculate(*bigNum* *a*, *bigNum* *b*);

void printBigNum(*bigNum* *num*);

// Hàm cộng 2 số nguyên lớn

*bigNum* add(*bigNum* *a*, *bigNum* *b*) {

    // Xử lý dấu

    if (*a*.sign == *b*.sign) {

*string* result;

        int carry = 0;

        int lenA = *a*.num.length(), lenB = *b*.num.length();

        int i = lenA - 1, j = lenB - 1;

        while (i >= 0 || j >= 0 || carry) {

            int sum = carry;

            if (i >= 0) sum += *a*.num[i] - '0';

            if (j >= 0) sum += *b*.num[j] - '0';

            carry = sum / 10;

            result.push\_back(sum % 10 + '0');

            i--; j--;

        }

        reverse(result.begin(), result.end());

        return {*a*.sign, result};

    } else {

        // Nếu dấu khác nhau thì trừ

        if (*a*.sign == '1') {

*a*.sign = '0';  // Đổi dấu a và cộng

            return subtract(*b*, *a*);

        } else {

*b*.sign = '0';  // Đổi dấu b và cộng

            return subtract(*a*, *b*);

        }

    }

}

// Hàm trừ 2 số nguyên lớn

*bigNum* subtract(*bigNum* *a*, *bigNum* *b*) {

    // Đảm bảo a >= b

    if (*a*.sign != *b*.sign) {

*b*.sign = *a*.sign;  // Đổi dấu và cộng

        return add(*a*, *b*);

    }

*string* result;

    int borrow = 0;

    int lenA = *a*.num.length(), lenB = *b*.num.length();

    int i = lenA - 1, j = lenB - 1;

    while (i >= 0 || j >= 0 || borrow) {

        int sub = (*a*.num[i] - '0') - (j >= 0 ? *b*.num[j] - '0' : 0) - borrow;

        borrow = (sub < 0) ? 1 : 0;

        if (borrow) sub += 10;

        result.push\_back(sub + '0');

        i--; j--;

    }

    reverse(result.begin(), result.end());

    return {*a*.sign, result};

}

// Hàm nhân 2 số nguyên lớn

*bigNum* multiply(*bigNum* *a*, *bigNum* *b*) {

*string* result(*a*.num.length() + *b*.num.length(), '0');

    int sign = (*a*.sign == *b*.sign) ? 1 : 0;

    for (int i = *a*.num.length() - 1; i >= 0; i--) {

        int carry = 0;

        for (int j = *b*.num.length() - 1; j >= 0; j--) {

            int mul = (*a*.num[i] - '0') \* (*b*.num[j] - '0') + carry + (result[i + j + 1] - '0');

            carry = mul / 10;

            result[i + j + 1] = mul % 10 + '0';

        }

        result[i] = carry + '0';

    }

    // Remove leading zeros

    int start = 0;

    while (start < result.length() && result[start] == '0') start++;

    result = result.substr(start);

    if (result.empty()) result = "0";  // Nếu kết quả là 0

    return {sign ? '1' : '0', result};

}

// Hàm tính biểu thức ab - 3a + 4b

*bigNum* calculate(*bigNum* *a*, *bigNum* *b*) {

*bigNum* ab = multiply(*a*, *b*); // Tính ab

*bigNum* threeA = multiply({'1', "3"}, *a*); // 3a

*bigNum* fourB = multiply({'1', "4"}, *b*); // 4b

*bigNum* result = subtract(ab, threeA); // Tính ab - 3a

    result = add(result, fourB);  // Tính (ab - 3a) + 4b

    return result;

}

// Hàm in ra số nguyên lớn theo định dạng

void printBigNum(*bigNum* *num*) {

    cout << *num*.sign << *num*.num << endl;

}

int main() {

*string* a\_str, b\_str;

    cin >> a\_str >> b\_str;

*bigNum* a = {a\_str[0], a\_str.substr(1)};

*bigNum* b = {b\_str[0], b\_str.substr(1)};

*bigNum* result = calculate(a, b);

    printBigNum(result);

    return 0;

}

/\*

0121807015

1347227347

\*/

A screen shot of a computer

Description automatically generated