Contents

[Bài thực hành số 2 – Tuần 34 2](#_Toc136445854)

[**Bài tập 2.1.**Viết hàm tính độ dài cạnh huyền của tam giác theo độ hai cạnh góc vuông. 2](#_Toc136445855)

[**Bài tập 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](#_Toc136445856)

[**Bài tập 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. 7](#_Toc136445857)

[**Bài tập 2.4.**Viết các hàm tính lập phương của số nguyên và số thực. 10](#_Toc136445858)

[**Bài tập 2.5.** Viết các toán tử tính tổng, hiệu, tích và thương của hai số phức 13](#_Toc136445859)

[**Bài tập 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. 17](#_Toc136445860)

[**Bài tập 2.7.**Viết hàm tính tổng các phần tử trong hai mảng. 21](#_Toc136445861)

[**Bài tập 2.8.**Viết hàm so sánh cho thuật toán sắp xếp. 24](#_Toc136445862)

[**Bài tập 2.9.**Viết chương trình in ra tất cả các dãy con của một dãy cho trước. 27](#_Toc136445863)

[**Bài tập 2.10.** Dưới đây cung cấp đoạn code đơn giản để tính tích của hai ma trận cỡ NxN theo công thức trực tiếp. 35](#_Toc136445864)

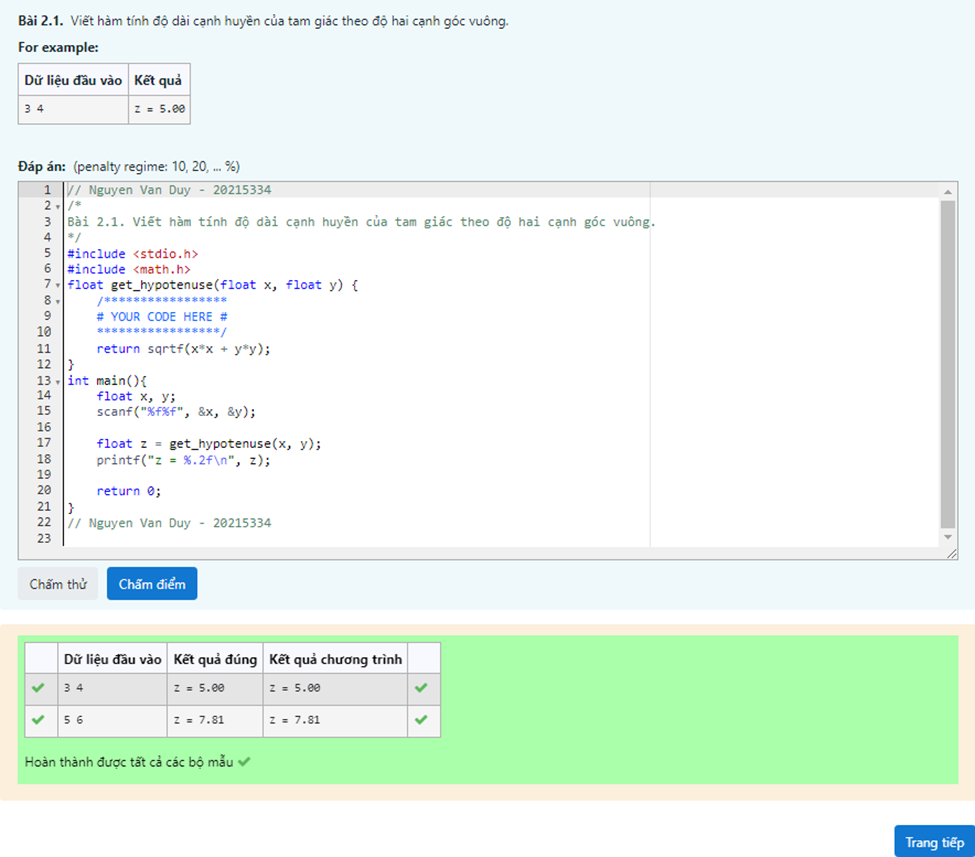
[**Bài tập 2.11.** Cho 2 đa thức A(x) và B(x) tương ứng có bậc N và M. Hãy tính ma trận tích C(x) = A(x) \* B(x) có bậc N+M−1 42](#_Toc136445865)

[**Bài tập 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. 49](#_Toc136445866)

[**Bài tập 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: 54](#_Toc136445867)

# Bài thực hành số 2 – Tuần 34

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



// Nguyen Van Duy - 20215334

/\*

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

\*/

#include <stdio.h>

#include <math.h>

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return sqrtf(x\*x + y\*y);

}

int main(){

float x, y;

scanf("%f%f", &x, &y);

float z = get\_hypotenuse(x, y);

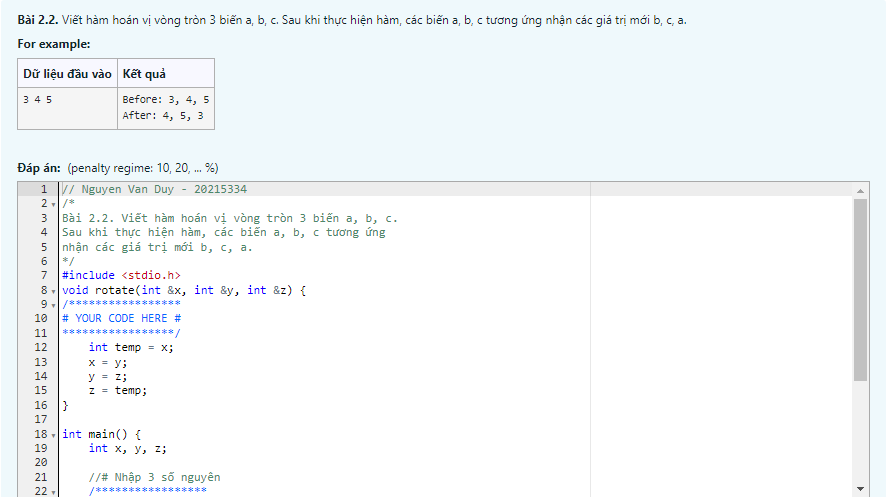
printf("z = %.2f\n", z);

return 0;

}

// Nguyen Van Duy - 20215334

## **Bài tập 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.



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Trang web

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

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.

\*/

#include <stdio.h>

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int temp = x;

x = y;

y = z;

z = temp;

}

int main() {

int x, y, z;

//# Nhập 3 số nguyên

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

scanf("%d%d%d", &x, &y, &z);

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

rotate(x, y, z);

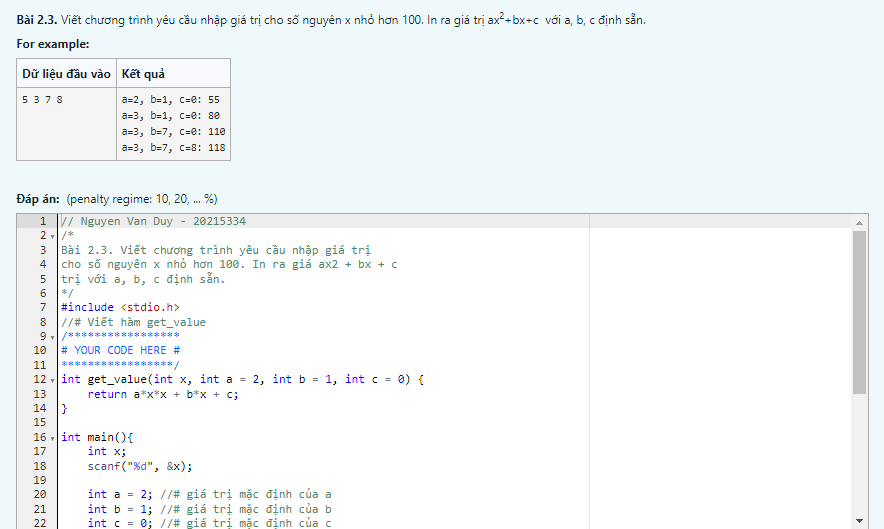
printf("After: %d, %d, %d\n", x, y, z);

return 0;

}

// Nguyen Van Duy - 20215334

## **Bài tập 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.



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Trang web

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

Bài 2.3. Viết chương trình yêu cầu nhập giá trị

cho số nguyên x nhỏ hơn 100. In ra giá ax2 + bx + c

trị với a, b, c định sẵn.

\*/

#include <stdio.h>

//# Viết hàm get\_value

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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; //# giá trị mặc định của a

int b = 1; //# giá trị mặc định của b

int c = 0; //# giá trị mặc định của c

//# Nhập 3 số nguyên a, b, c từ bàn phím

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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));

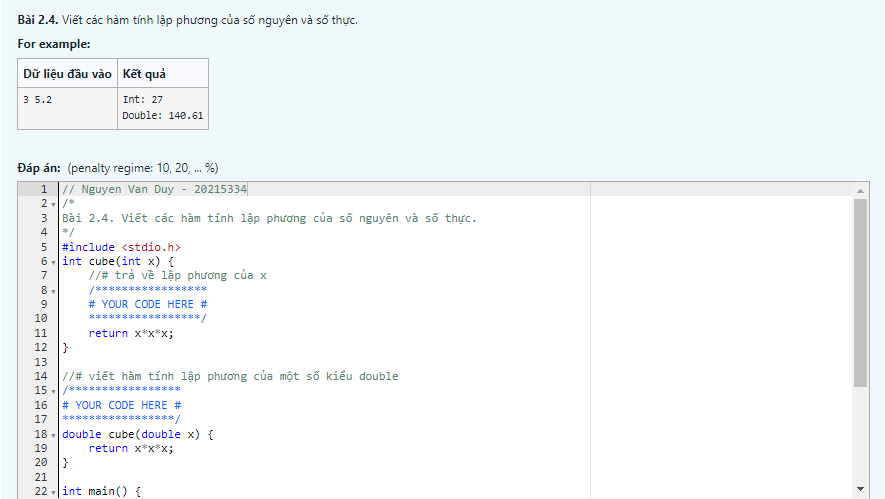
printf("a=%d, b=%d, c=%d: %d\n", a, b, c, get\_value(x, a, b, c));

return 0;

}

// Nguyen Van Duy - 20215334

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



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

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

\*/

#include <stdio.h>

int cube(int x) {

//# trả về lập phương của x

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return x\*x\*x;

}

//# viết hàm tính lập phương của một số kiểu double

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double cube(double x) {

return x\*x\*x;

}

int main() {

int n;

double f;

scanf("%d %lf", &n, &f);

printf("Int: %d\n", cube(n));

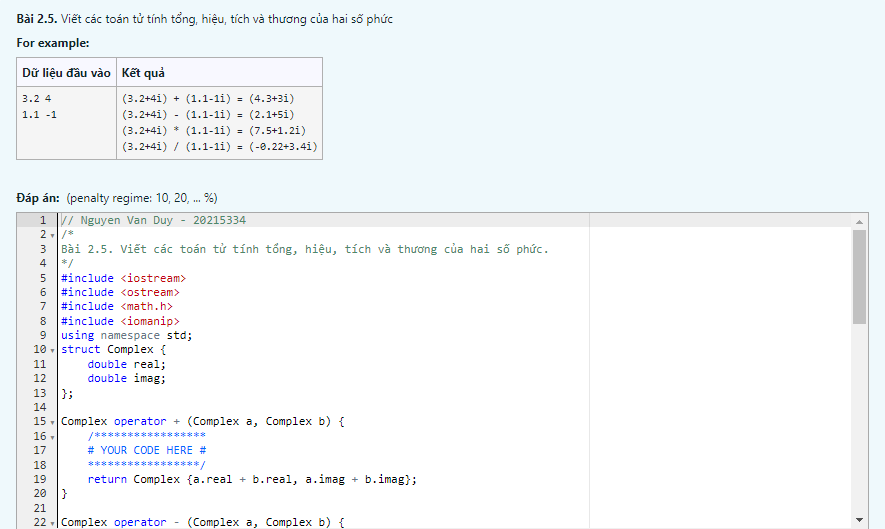
printf("Double: %.2lf\n", cube(f));

return 0;

}

// Nguyen Van Duy - 20215334

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



Ảnh có chứa văn bản, ảnh chụp màn hình, số, phần mềm

Mô tả được tạo tự động

Ảnh có chứa văn bản, phần mềm, Phông chữ, Trang web

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

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

\*/

#include <iostream>

#include <ostream>

#include <math.h>

#include <iomanip>

using namespace std;

struct Complex {

double real;

double imag;

};

Complex operator + (Complex a, Complex b) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return Complex {a.real + b.real, a.imag + b.imag};

}

Complex operator - (Complex a, Complex b) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return Complex {a.real - b.real, a.imag - b.imag};

}

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return Complex {a.real \* b.real - a.imag \* b.imag, a.imag \* b.real + a.real \* b.imag};

}

Complex operator / (Complex a, Complex b) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double temp = b.real \* b.real + b.imag \* b.imag;

return Complex {(a.real \* b.real + a.imag \* b.imag) / temp, (a.imag \* b.real - a.real \* b.imag) / temp};

}

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;

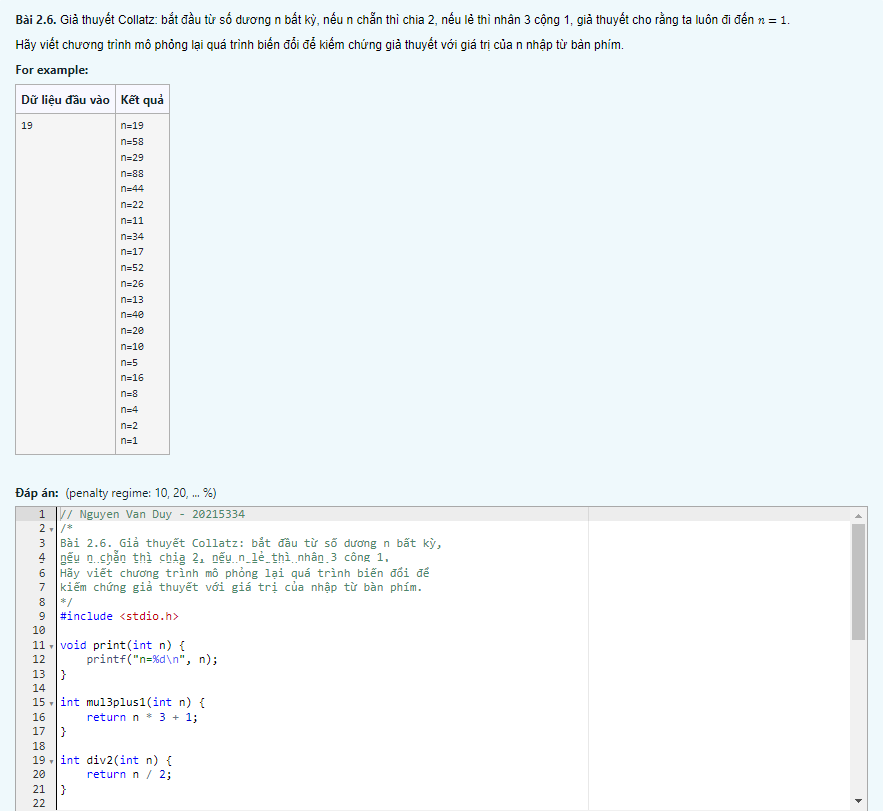
return 0;

}

// Nguyen Van Duy - 20215334

## **Bài tập 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.



Ảnh có chứa văn bản, ảnh chụp màn hình, hàng, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

Bài 2.6. Giả thuyết Collatz: bắt đầu từ số dương n bất kỳ,

nếu n chẵn thì chia 2, nếu n 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 nhập từ bàn phím.

\*/

#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 số cho các con trỏ hàm odd, even và output

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

{

(\*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;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int n;

scanf("%d", &n);

simulate(n, odd, even, print);

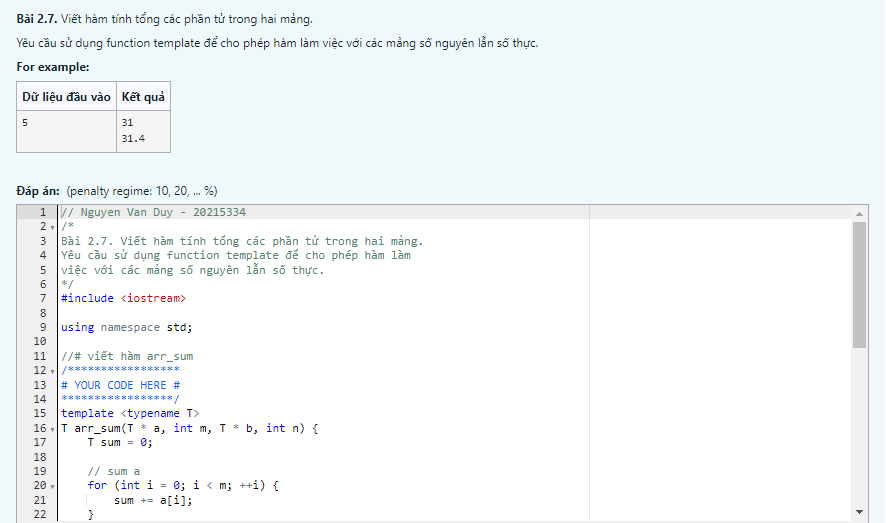
return 0;

}

// Nguyen Van Duy - 20215334

## **Bài tập 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.

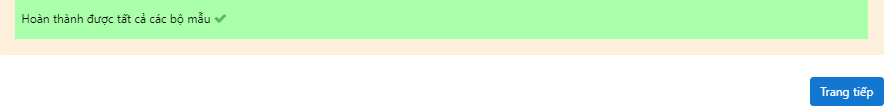


Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, hàng, số

Mô tả được tạo tự động



// Nguyen Van Duy - 20215334

/\*

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.

\*/

#include <iostream>

using namespace std;

//# viết hàm arr\_sum

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename T>

T arr\_sum(T \* a, int m, T \* b, int n) {

T sum = 0;

// sum a

for (int i = 0; i < m; ++i) {

sum += a[i];

}

// sum b

for (int i = 0; i < n; ++i) {

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;

}

// Nguyen Van Duy - 20215334

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

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, số

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

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

\*/

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

using namespace std;

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 trong a theo tổng các phần tử giảm dần

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sort(a.begin(), a.end(), [](vector<int> a, vector<int> b) -> bool {

return accumulate(a.begin(), a.end(), 0) > accumulate(b.begin(), b.end(), 0);

});

for (const auto &v : a) {

for (int it : v) {

cout << it << ' ';

}

cout << endl;

}

return 0;

}

// Nguyen Van Duy - 20215334

## **Bài tập 2.9.**Viết chương trình in ra tất cả các dãy con của một dãy cho trước.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, số

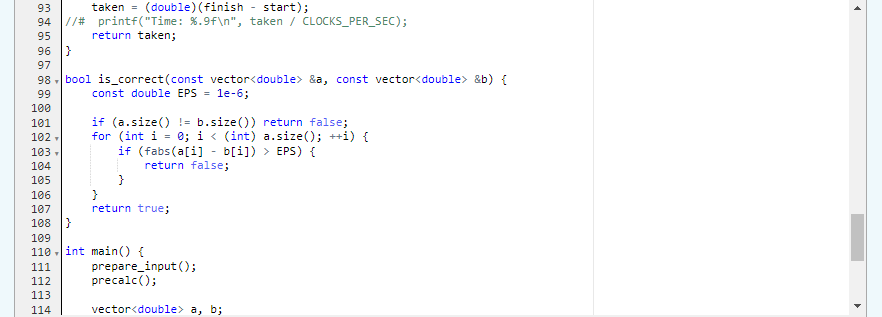
Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, số, Phông chữ

Mô tả được tạo tự động



Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, hàng

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

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.

\*/

#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;

}

}

//# BEGIN fast code

//# khai báo các biến phụ trợ cần thiết

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

const int TABLE\_SIZE = 40001;

const double TABLE\_MIN = -20;

const double TABLE\_MAX = 20;

const double TABLE\_STEP = 0.001;

double sigmoid\_table[TABLE\_SIZE];

//# hàm chuẩn bị dữ liệu

void precalc() {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

double x = TABLE\_MIN + i \* TABLE\_STEP;

sigmoid\_table[i] = 1.0 / (1.0 + exp(-x));

}

}

//# hàm tính sigmoid(x) nhanh sigmoid\_fast(x)

inline double sigmoid\_fast(double x) {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (x < TABLE\_MIN) return sigmoid\_table[0];

else if (x > TABLE\_MAX) return sigmoid\_table[TABLE\_SIZE-1];

else {

double i = (x - TABLE\_MIN) / TABLE\_STEP;

int index = (int) i;

return sigmoid\_table[index] + (sigmoid\_table[index+1] - sigmoid\_table[index]) \* (i - index);

}

}

//# END fast code

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 (int i = 0; i < (int) 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("Correct answer! Your code is faster at least 30%%!\n");

} else {

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

}

return 0;

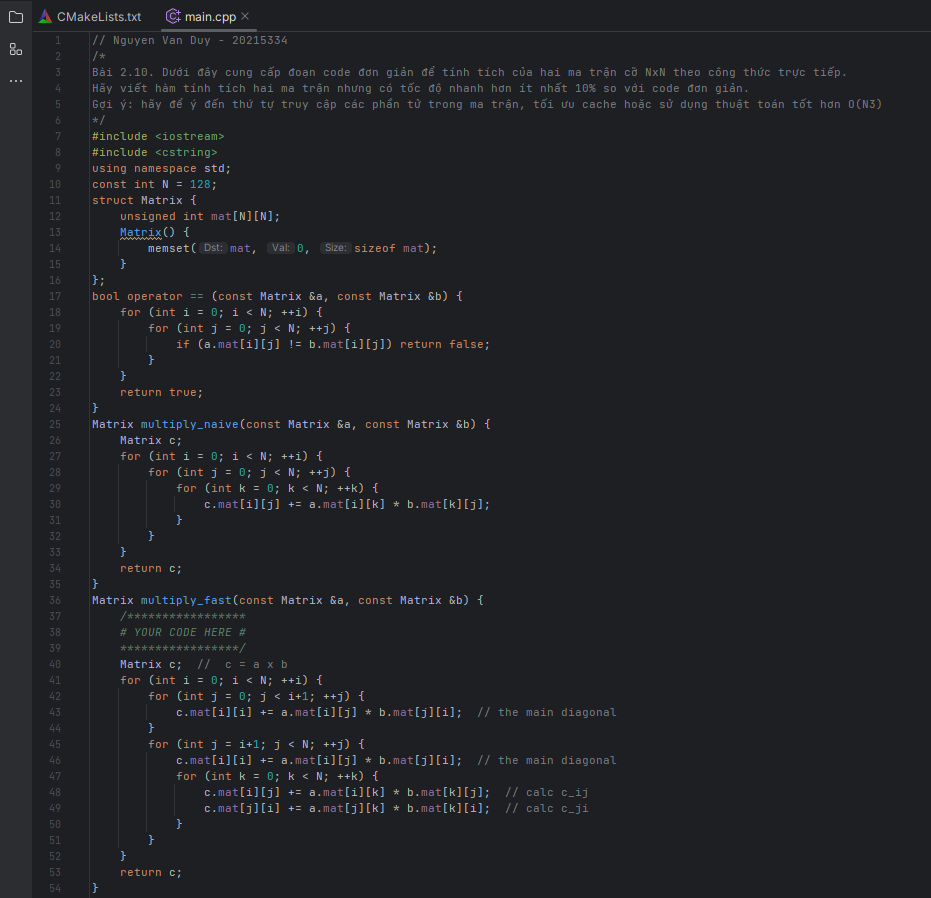
}

// Nguyen Van Duy - 20215334

## **Bài tập 2.10.** Dưới đây cung cấp đoạn code đơn giản để tính tích của hai ma trận cỡ NxN 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).



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

Bài 2.10. Dưới đây cung cấp đoạn code đơn giản để tính tích của hai ma trận cỡ NxN 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)

\*/

#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;

}

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;

}

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# YOUR CODE HERE #

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Matrix c; // c = a x b

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

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

c.mat[i][i] += a.mat[i][j] \* b.mat[j][i]; // the main diagonal

}

for (int j = i+1; j < N; ++j) {

c.mat[i][i] += a.mat[i][j] \* b.mat[j][i]; // the main diagonal

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

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

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

}

}

}

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");

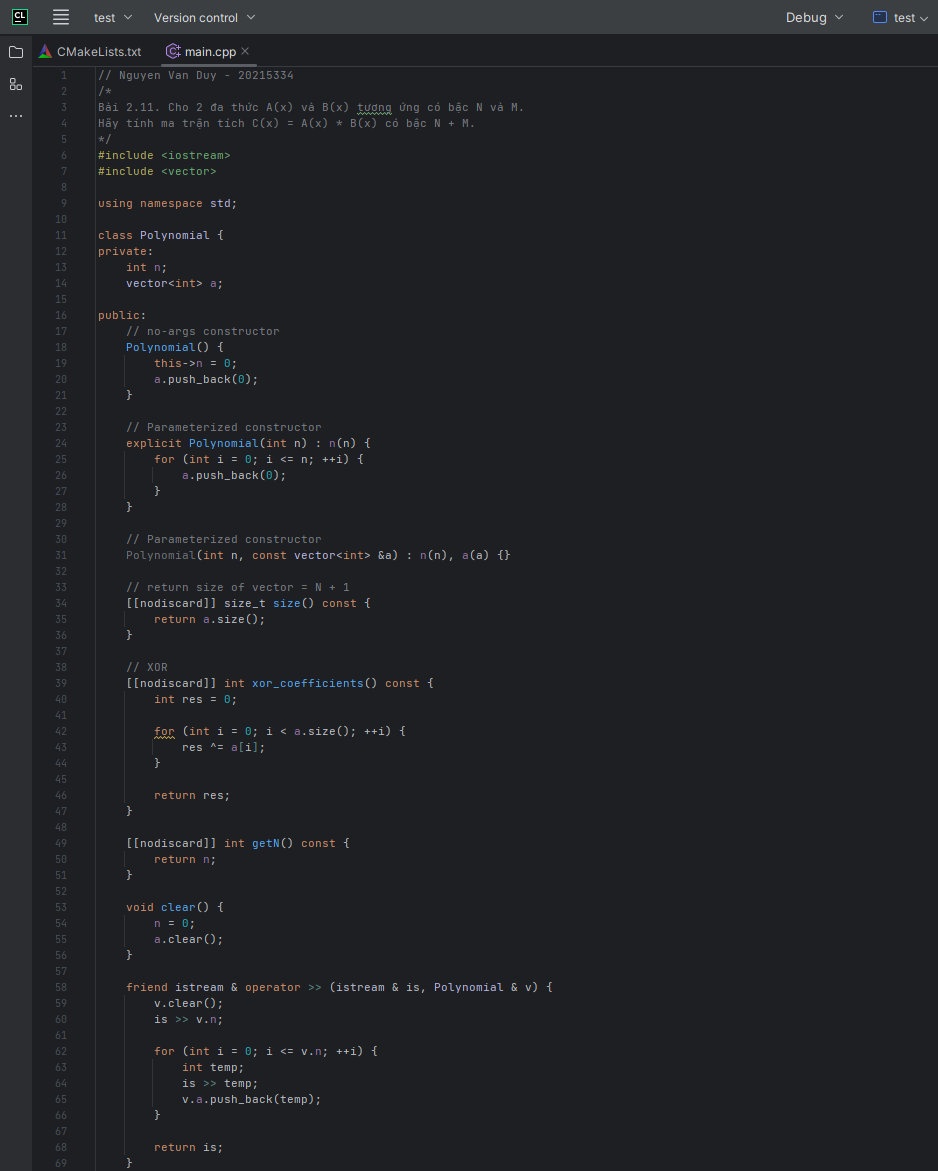
}

return 0;

}

// Nguyen Van Duy – 20215334

## **Bài tập 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



Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

// Nguyen Van Duy - 20215334

/\*

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.

\*/

#include <iostream>

#include <vector>

using namespace std;

class Polynomial {

private:

int n;

vector<int> a;

public:

// no-args constructor

Polynomial() {

this->n = 0;

a.push\_back(0);

}

// Parameterized constructor

explicit Polynomial(int n) : n(n) {

for (int i = 0; i <= n; ++i) {

a.push\_back(0);

}

}

// Parameterized constructor

Polynomial(int n, const vector<int> &a) : n(n), a(a) {}

// return size of vector = N + 1

[[nodiscard]] size\_t size() const {

return a.size();

}

// XOR

[[nodiscard]] int xor\_coefficients() const {

int res = 0;

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

res ^= a[i];

}

return res;

}

[[nodiscard]] int getN() const {

return n;

}

void clear() {

n = 0;

a.clear();

}

friend istream & operator >> (istream & is, Polynomial & v) {

v.clear();

is >> v.n;

for (int i = 0; i <= v.n; ++i) {

int temp;

is >> temp;

v.a.push\_back(temp);

}

return is;

}

friend ostream & operator << (ostream & os, Polynomial v) {

for (int i = 0; i < v.n; ++i) {

os << v.a[i] << " ";

}

os << v.a[v.n];

return os;

}

// multiplying two polynomials

Polynomial operator \* (const Polynomial & v) const {

Polynomial res(this->n + v.n);

for (int i = 0; i <= res.n; ++i) {

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

if (j < this->size() && i-j < v.size()) {

res.a[res.getN() - i] += this->a[this->getN()-j] \* v.a[v.getN()-(i-j)];

}

}

}

return res;

}

};

int main() {

Polynomial a, b;

// enter 2 polynomials

cin >> a >> b;

// XOR of the coefficients of the polynomial

cout << (a\*b).xor\_coefficients() << endl;

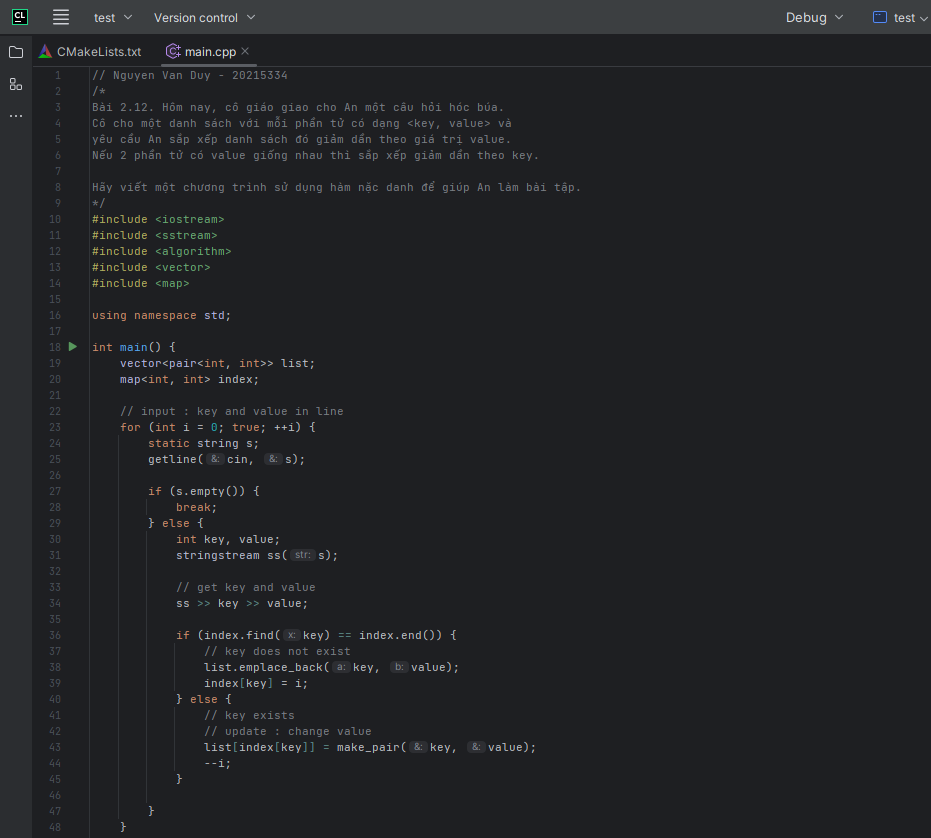
return 0;

}

// Nguyen Van Duy - 20215334

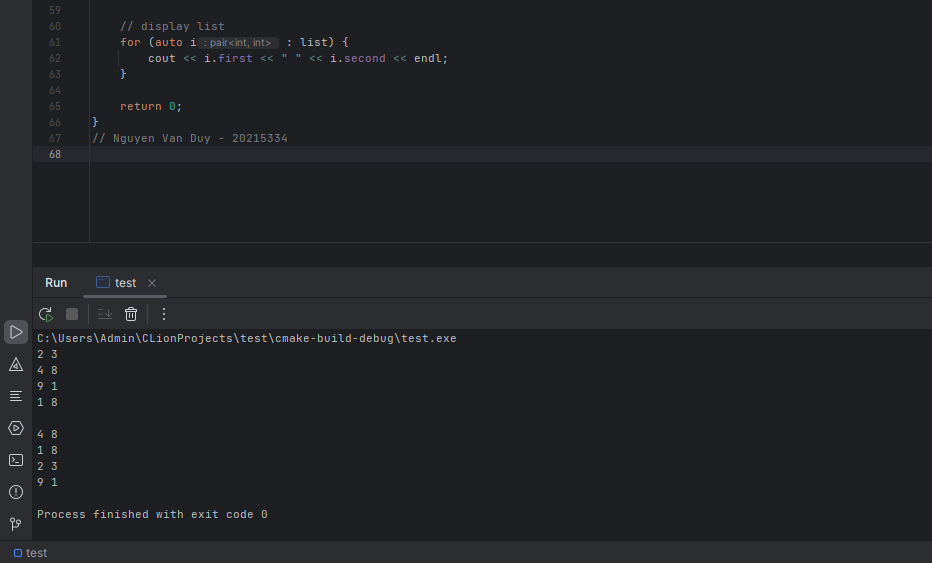
## **Bài tập 2.12.** Hôm nay, cô giáo giao cho An một câu hỏi hóc búa. Cô cho một danh sách với mỗi phần tử có dạng <key, value> và yêu cầu An sắp xếp danh sách đó giảm dần theo giá trị value. Nếu 2 phần tử có value giống nhau thì sắp xếp giảm dần theo key.

Hãy viết một chương trình sử dụng hàm nặc danh để giúp An làm bài tập.



Ảnh có chứa văn bản, ảnh chụp màn hình

Mô tả được tạo tự động



// Nguyen Van Duy - 20215334

/\*

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.

\*/

#include <iostream>

#include <sstream>

#include <algorithm>

#include <vector>

#include <map>

using namespace std;

int main() {

vector<pair<int, int>> list;

map<int, int> index;

// input : key and value in line

for (int i = 0; true; ++i) {

static string s;

getline(cin, s);

if (s.empty()) {

break;

} else {

int key, value;

stringstream ss(s);

// get key and value

ss >> key >> value;

if (index.find(key) == index.end()) {

// key does not exist

list.emplace\_back(key, value);

index[key] = i;

} else {

// key exists

// update : change value

list[index[key]] = make\_pair(key, value);

--i;

}

}

}

// sort by value

sort(list.begin(), list.end(), [](pair<int, int> a, pair<int, int> b) -> bool {

if (a.second > b.second) return true;

if (a.second < b.second) return false;

// 2 values are equal

// sort by key

return a.first > b.first;

});

// display list

for (auto i : list) {

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

}

return 0;

}

// Nguyen Van Duy - 20215334

## **Bài tập 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:

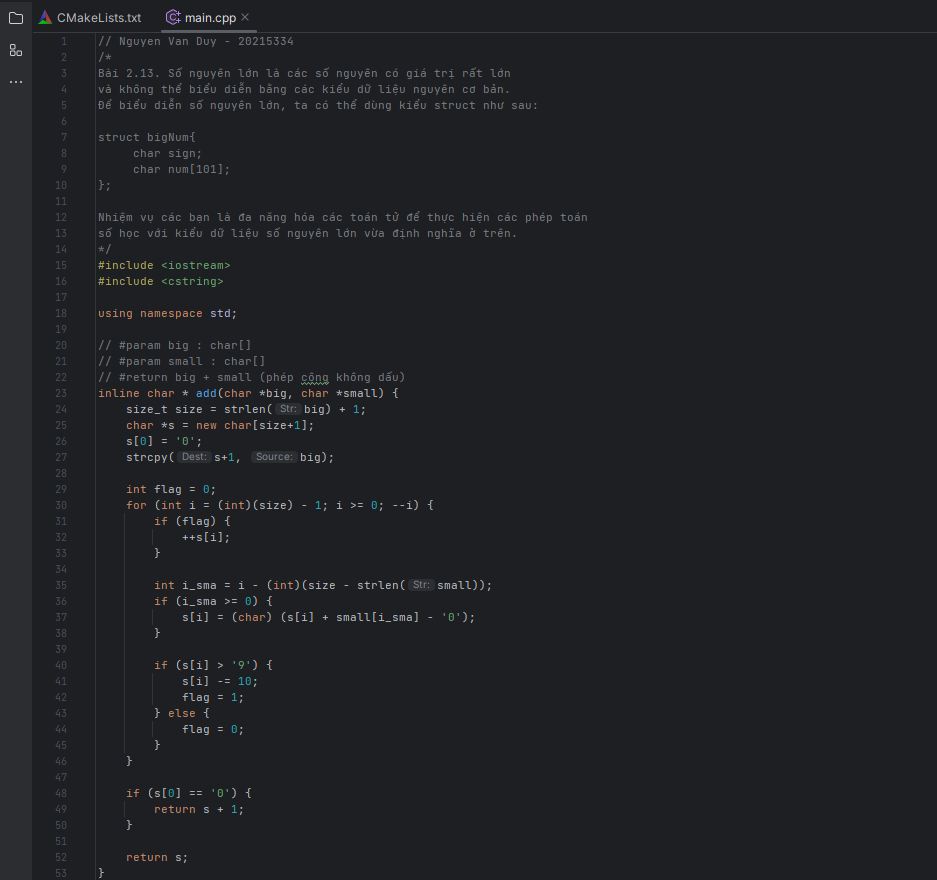
struct bigNum{

char sign;

char num[101];

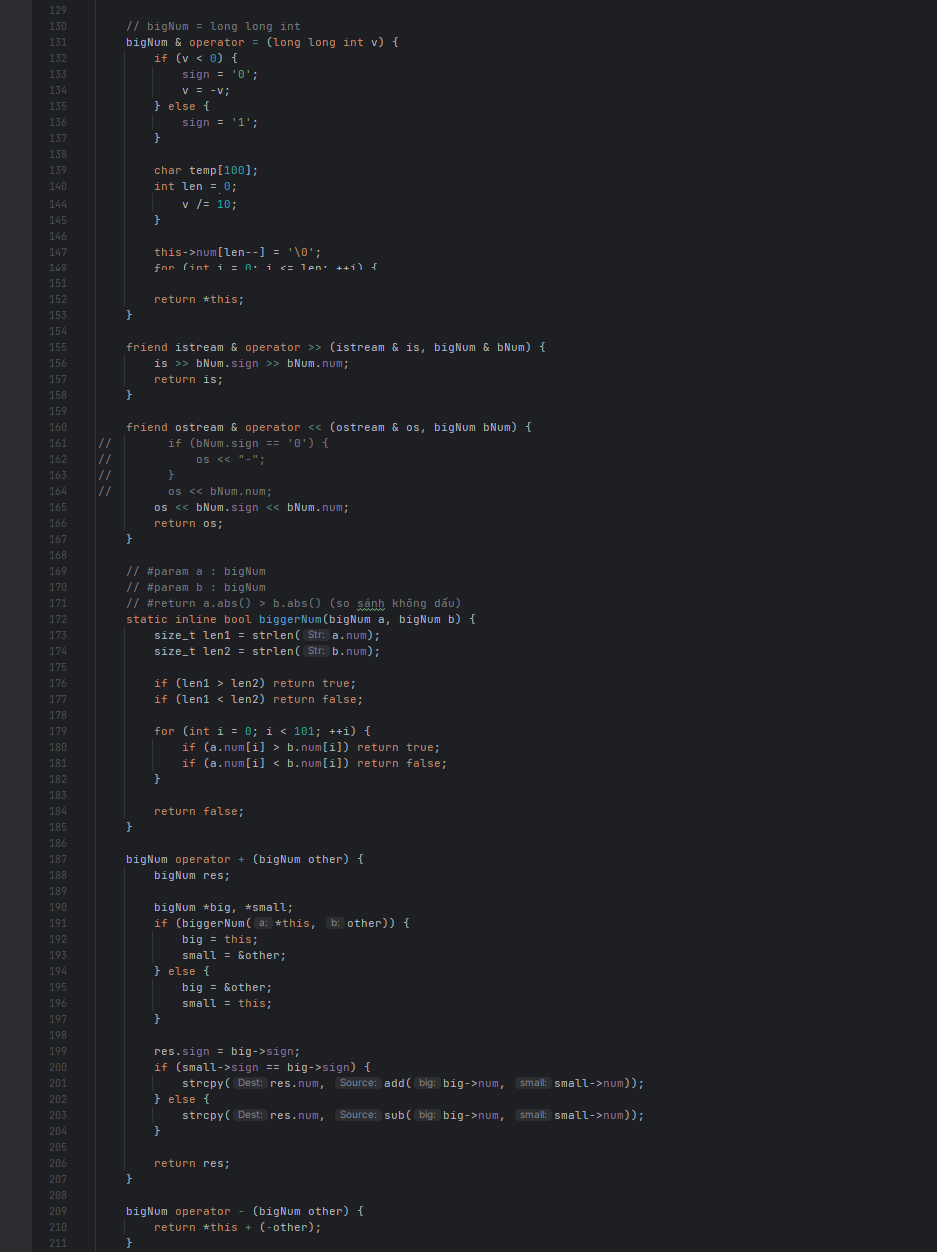
};

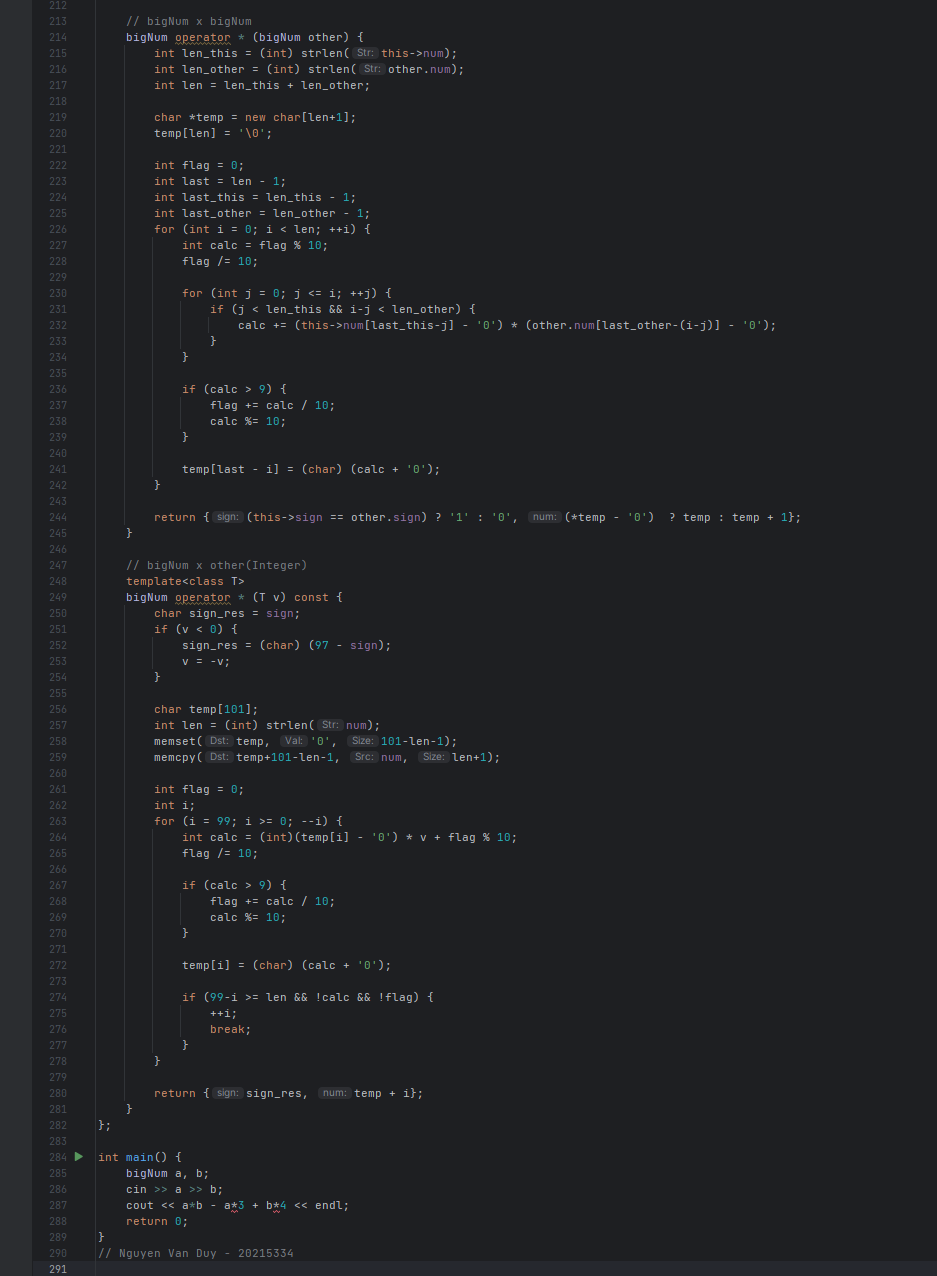
Nhiệm vụ các bạn là đa năng hóa các toán tử để thực hiện các phép toán số học với kiểu dữ liệu số nguyên lớn vừa định nghĩa ở trên.

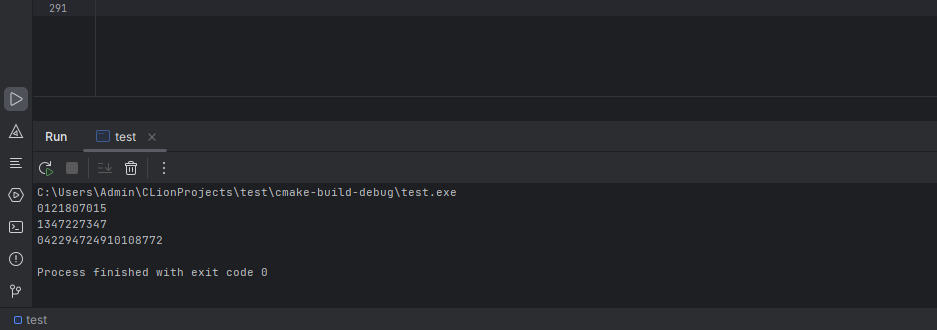


Ảnh có chứa văn bản, ảnh chụp màn hình

Mô tả được tạo tự động







// Nguyen Van Duy - 20215334

/\*

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.

\*/

#include <iostream>

#include <cstring>

using namespace std;

// #param big : char[]

// #param small : char[]

// #return big + small (phép cộng không dấu)

inline char \* add(char \*big, char \*small) {

size\_t size = strlen(big) + 1;

char \*s = new char[size+1];

s[0] = '0';

strcpy(s+1, big);

int flag = 0;

for (int i = (int)(size) - 1; i >= 0; --i) {

if (flag) {

++s[i];

}

int i\_sma = i - (int)(size - strlen(small));

if (i\_sma >= 0) {

s[i] = (char) (s[i] + small[i\_sma] - '0');

}

if (s[i] > '9') {

s[i] -= 10;

flag = 1;

} else {

flag = 0;

}

}

if (s[0] == '0') {

return s + 1;

}

return s;

}

// #param big : char[] --> số lớn hơn

// #param small : char[] --> số nhỏ hơn

// #return big - small (phép trừ không dấu)

inline char \* sub(char \*big, char \*small) {

size\_t size = strlen(big);

char \*s = new char[size+1];

strcpy(s, big);

int flag = 0;

for (int i = (int)(size) - 1; i >= 0; --i) {

if (flag) {

--s[i];

}

int i\_sma = i - (int)(size - strlen(small));

if (i\_sma >= 0) {

s[i] = (char) (s[i] - small[i\_sma] + '0');

}

if (s[i] < '0') {

s[i] += 10;

flag = -1;

} else {

flag = 0;

}

}

return s;

}

struct bigNum{

char sign;

char num[101]{};

// no-args constructor

bigNum() {

sign = '1';

num[0] = '0';

num[1] = '\0';

}

// Parameterized constructor

bigNum(char sign, const char \*num) {

this->sign = sign;

for (int i = 0; i < 101; ++i) {

this->num[i] = num[i];

}

}

// Parameterized constructor

bigNum(long long v) {

\*this = v;

}

// absolute value

bigNum abs() const {

bigNum res = \*this;

res.sign = '1';

return res;

}

// Opposite number

bigNum operator - () const {

bigNum res = \*this;

res.sign = (char) (97 - sign);

return res;

}

// bigNum = bigNum

bigNum & operator = (const bigNum v) {

sign = v.sign;

memcpy(num, v.num, 101);

return \*this;

}

// bigNum = long long int

bigNum & operator = (long long int v) {

if (v < 0) {

sign = '0';

v = -v;

} else {

sign = '1';

}

char temp[100];

int len = 0;

for (int i = 0; i < 100 && v; ++i) {

temp[i] = (char) (v % 10 + '0');

++len;

v /= 10;

}

this->num[len--] = '\0';

for (int i = 0; i <= len; ++i) {

num[len-i] = temp[i];

}

return \*this;

}

friend istream & operator >> (istream & is, bigNum & bNum) {

is >> bNum.sign >> bNum.num;

return is;

}

friend ostream & operator << (ostream & os, bigNum bNum) {

// if (bNum.sign == '0') {

// os << "-";

// }

// os << bNum.num;

os << bNum.sign << bNum.num;

return os;

}

// #param a : bigNum

// #param b : bigNum

// #return a.abs() > b.abs() (so sánh không dấu)

static inline bool biggerNum(bigNum a, bigNum b) {

size\_t len1 = strlen(a.num);

size\_t len2 = strlen(b.num);

if (len1 > len2) return true;

if (len1 < len2) return false;

for (int i = 0; i < 101; ++i) {

if (a.num[i] > b.num[i]) return true;

if (a.num[i] < b.num[i]) return false;

}

return false;

}

bigNum operator + (bigNum other) {

bigNum res;

bigNum \*big, \*small;

if (biggerNum(\*this, other)) {

big = this;

small = &other;

} else {

big = &other;

small = this;

}

res.sign = big->sign;

if (small->sign == big->sign) {

strcpy(res.num, add(big->num, small->num));

} else {

strcpy(res.num, sub(big->num, small->num));

}

return res;

}

bigNum operator - (bigNum other) {

return \*this + (-other);

}

// bigNum x bigNum

bigNum operator \* (bigNum other) {

int len\_this = (int) strlen(this->num);

int len\_other = (int) strlen(other.num);

int len = len\_this + len\_other;

char \*temp = new char[len+1];

temp[len] = '\0';

int flag = 0;

int last = len - 1;

int last\_this = len\_this - 1;

int last\_other = len\_other - 1;

for (int i = 0; i < len; ++i) {

int calc = flag % 10;

flag /= 10;

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

if (j < len\_this && i-j < len\_other) {

calc += (this->num[last\_this-j] - '0') \* (other.num[last\_other-(i-j)] - '0');

}

}

if (calc > 9) {

flag += calc / 10;

calc %= 10;

}

temp[last - i] = (char) (calc + '0');

}

return {(this->sign == other.sign) ? '1' : '0', (\*temp - '0') ? temp : temp + 1};

}

// bigNum x other(Integer)

template<class T>

bigNum operator \* (T v) const {

char sign\_res = sign;

if (v < 0) {

sign\_res = (char) (97 - sign);

v = -v;

}

char temp[101];

int len = (int) strlen(num);

memset(temp, '0', 101-len-1);

memcpy(temp+101-len-1, num, len+1);

int flag = 0;

int i;

for (i = 99; i >= 0; --i) {

int calc = (int)(temp[i] - '0') \* v + flag % 10;

flag /= 10;

if (calc > 9) {

flag += calc / 10;

calc %= 10;

}

temp[i] = (char) (calc + '0');

if (99-i >= len && !calc && !flag) {

++i;

break;

}

}

return {sign\_res, temp + i};

}

};

int main() {

bigNum a, b;

cin >> a >> b;

cout << a\*b - a\*3 + b\*4 << endl;

return 0;

}

// Nguyen Van Duy - 20215334