Contents

[Danh mục hình ảnh 2](#__RefHeading___Toc349_2734074527)

[Bài thực hành số 2: Hàm và tối ưu mã nguồn 3](#__RefHeading___Toc351_2734074527)

[Phần 1. Thực hành về hàm 3](#__RefHeading___Toc353_2734074527)

[1. Bài tập 1: Truyền tham trị 3](#__RefHeading___Toc355_2734074527)

[2. Bài tập 2: Truyền tham chiếu 4](#__RefHeading___Toc357_2734074527)

[3. Bài tập 3: tham số ngầm định 5](#__RefHeading___Toc359_2734074527)

[1. Bài tập 4: đa năng hóa hàm 6](#__RefHeading___Toc363_2734074527)

[2. Bài tập 5: đa năng hóa toán từ 7](#__RefHeading___Toc365_2734074527)

[3. Bài tập 6: con trỏ hàm 9](#__RefHeading___Toc367_2734074527)

[1. Bài tập 7: Khái quát hóa hàm 10](#__RefHeading___Toc371_2734074527)

[2. Bài tập 8: Sắp xếp 12](#__RefHeading___Toc373_2734074527)

[Phần 2. Thực hành về tối ưu mã nguồn 13](#__RefHeading___Toc375_2734074527)

[1. Bài tập 9: Tính hàm sigmoid 13](#__RefHeading___Toc377_2734074527)

[2. Bài tập 10: Tính tích hai ma trận vuông 14](#__RefHeading___Toc379_2734074527)

[Phần 3. Bài tập về nhà 17](#__RefHeading___Toc843_1610690254)

[1. Bài tập 11: Tính tích hai đa thức 17](#__RefHeading___Toc845_1610690254)

[2. Bài tập 12: Map sort 18](#__RefHeading___Toc847_1610690254)

[3. Bài tập 13: Big integer 20](#__RefHeading___Toc849_1610690254)

# Danh mục hình ảnh

[Hình 1 Bài 1.1 Truyền tham trị, tham chiếu và tham số ngầm định 3](#Hình!0|sequence)

[Hình 2 Bài 1.2 Truyền tham trị, tham chiếu và tham số ngầm định 4](#Hình!1|sequence)

[Hình 3 Bài 1.3 Truyền tham trị, tham chiếu và tham số ngầm định 5](#Hình!2|sequence)

[Hình 4 Bài 1.4 Đa năng hóa hàm 6](#Hình!3|sequence)

[Hình 5 Bài 1.5 đa năng hóa toán từ 8](#Hình!4|sequence)

[Hình 6 Bài 1.6 con trỏ hàm 10](#Hình!5|sequence)

[Hình 7 Bài 1.7 Khái quát hóa hàm 11](#Hình!6|sequence)

[Hình 8 Bài 1.8 Biểu thức lamda và hàm nặc danh 13](#Hình!7|sequence)

[Hình 9 Bài 1.9 Tính hàm sigmoid 14](#Hình!8|sequence)

[Hình 10 Bài 1.10 Tính tích hai ma trận vuông 17](#Hình!9|sequence)

[Hình 11 Bài 1.11 Tính tích hai đa thức 18](#Hình!10|sequence)

[Hình 12 Bài 1.12 Map sort 19](#Hình!11|sequence)

[Hình 13 Bài 1.13 Big integer 20](#Hình!12|sequence)

# Bài thực hành số 2**:** **Hàm và tối ưu mã nguồn**

## Phần 1. Thực hành về hàm

### Bài tập 1: **Truyền tham trị**

**Tên file: MSSV-Hovaten\_Bai 1\_1.c**

*#include* <stdio.h>

*#include* <math.h>

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

*return* sqrt(pow(x,2) + pow(y,2));

}

int main(){

float x, y;

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

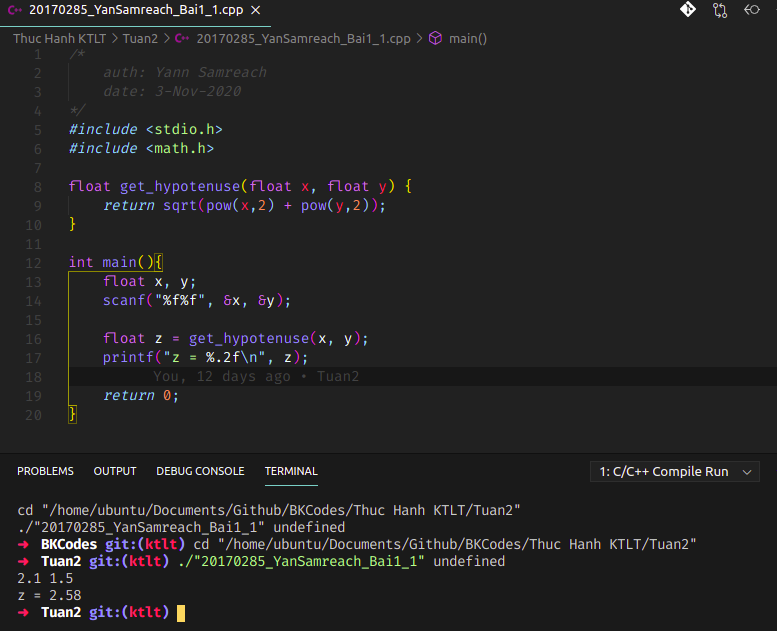
float z = get\_hypotenuse(x, y);

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

*return* 0;

}

Kết quả:



Hình 1 Bài 1.1 Truyền tham trị, tham chiếu và tham số ngầm định

### Bài tập 2: Truyền tham chiếu

*#include* <stdio.h>

*#include* <math.h>

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

*return* sqrt(pow(x,2) + pow(y,2));

}

int main(){

float x, y;

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

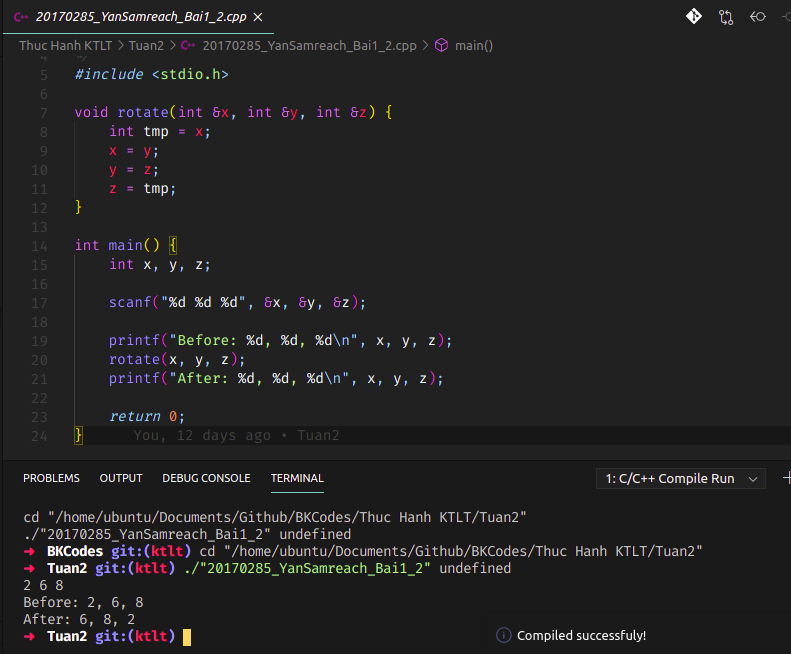
float z = get\_hypotenuse(x, y);

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

*return* 0;

}

Kết quả:



Hình 2 Bài 1.2 Truyền tham trị, tham chiếu và tham số ngầm định

### Bài tập 3: tham số ngầm định

*#include* <stdio.h>

*#include* <math.h>

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

*return* a\*pow(x,2) + 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*

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));

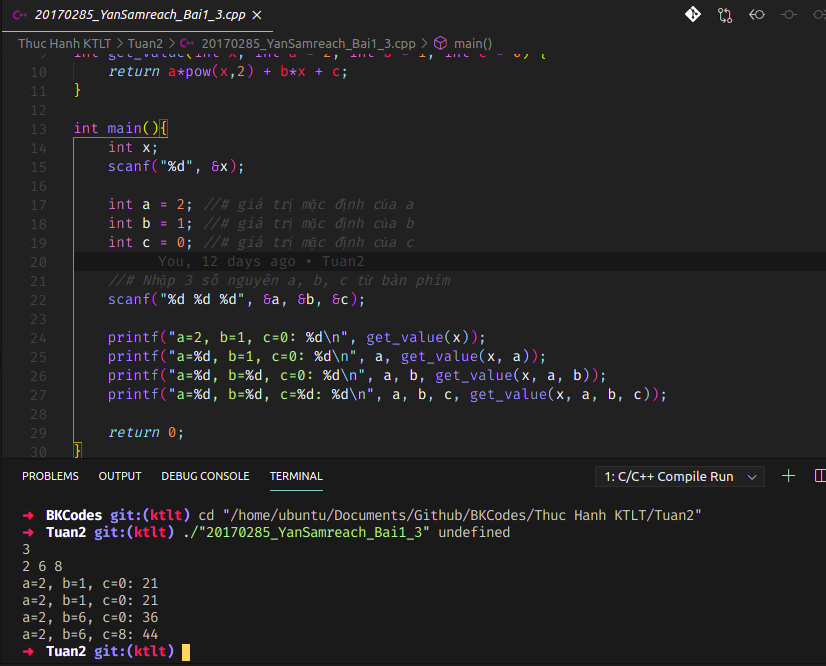
printf("a=%d, b=%d, c=0: %d\n", a, b, get\_value(x, a, b));

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

*return* 0;

}

Kết quả:



Hình 3 Bài 1.3 Truyền tham trị, tham chiếu và tham số ngầm định

### Bài tập 4: đa năng hóa hàm

*#include* <stdio.h>

int cube(int x) {

*return* x \* x \* x;

}

double cube(double x) {

*return* x \* x \* x;

}

int main() {

int n;

double f;

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

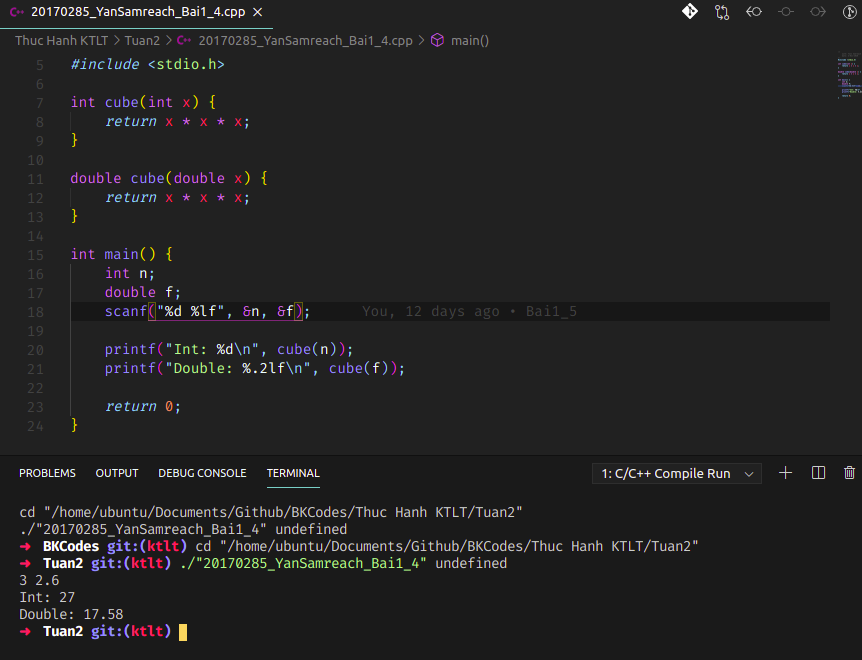
printf("Int: %d\n", cube(n));

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

*return* 0;

}

Kết quả:



Hình 4 Bài 1.4 Đa năng hóa hàm

### Bài tập 5: đa năng hóa toán từ

*#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 t;

t.real = a.real + b.real;

t.imag = a.imag + b.imag;

*return* t;

}

Complex operator - (Complex a, Complex b) {

Complex t;

t.real = a.real - b.real;

t.imag = a.imag - b.imag;

*return* t;

}

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

Complex tmp;

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

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

*return* tmp;

}

Complex operator / (Complex a, Complex b) {

Complex tmp;

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

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

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

*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;

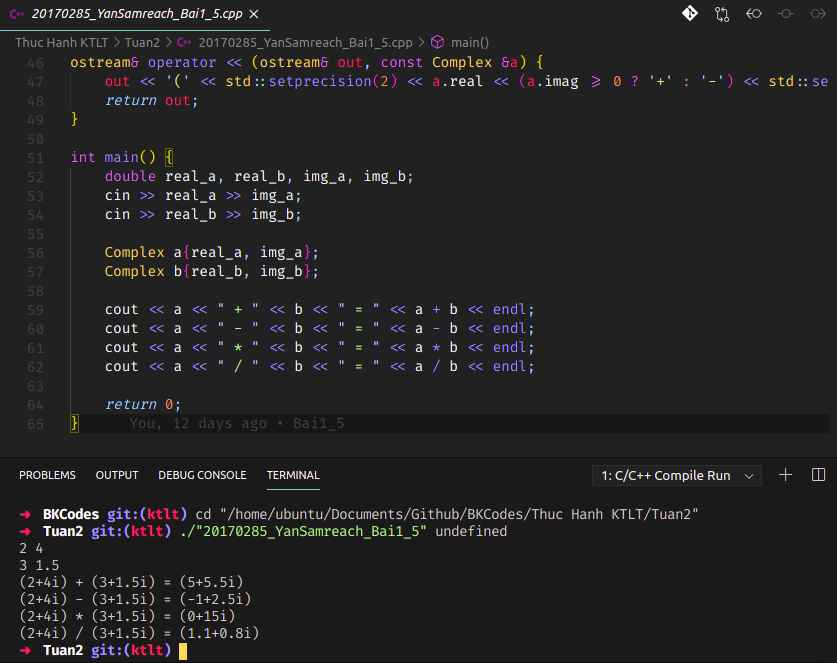
cout << a << " \* " << b << " = " << a \* b << endl;

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

*return* 0;

}

Kết quả:



Hình 5 Bài 1.5 đa năng hóa toán từ

### Bài tập 6: con trỏ hàm

*#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;

}

void simulate(int n, int (\*odd)(int) = mul3plus1, int (\*even)(int) = div2, void (\*output)(int) = print) {

(\*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;

int n;

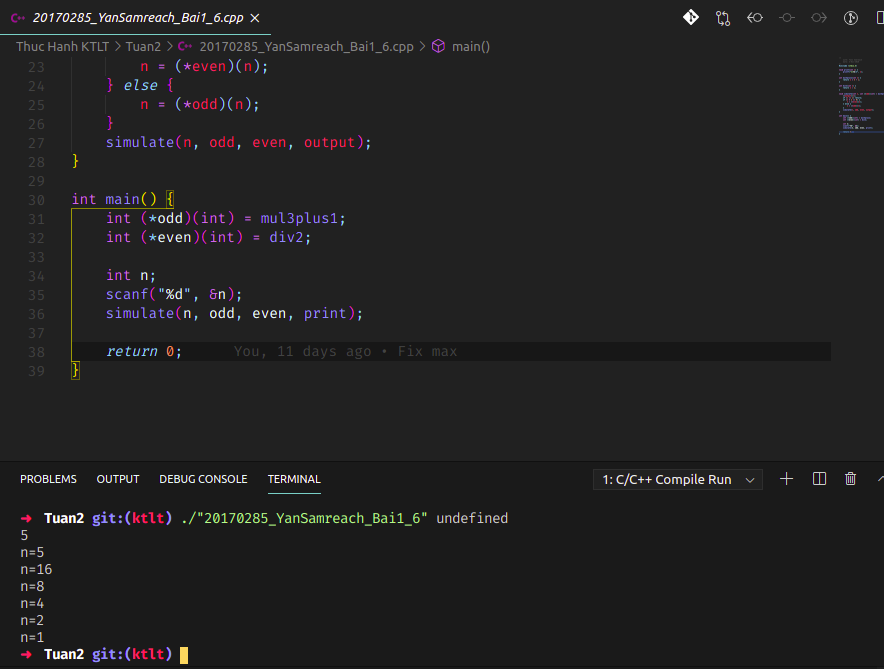
scanf("%d", &n);

simulate(n, odd, even, print);

*return* 0;

}

Kết quả:



Hình 6 Bài 1.6 con trỏ hàm

### Bài tập 7: Khái quát hóa hàm

*#include* <bits/stdc++.h>

using namespace std;

template <typename T>

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

T tmp = 0;

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

tmp += a[i];

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

tmp += b[i];

*return* tmp;

}

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};

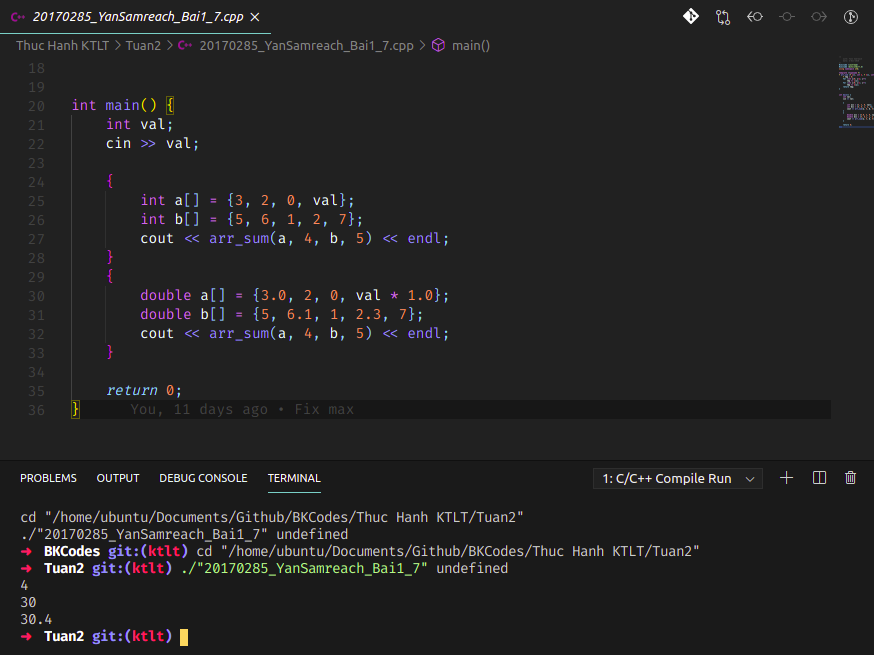
cout << arr\_sum(a, 4, b, 5) << endl;

}

*return* 0;

}

Kết quả:



Hình 7 Bài 1.7 Khái quát hóa hàm

### Bài tập 8: Sắp xếp

*#include* <iostream>

*#include* <vector>

*#include* <algorithm>

*#include* <numeric>

using namespace std;

bool compare(vector<int> a, vector<int> b){

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

}

int main() {

int val1, val2;

cin >> val1 >> val2;

vector< vector<int> > a = {

{1, 3, 7},

{2, 3, 4, val1},

{9, 8, 15},

{10, val2},

};

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

*for* (const auto &v : a) {

*for* (int it : v) {

cout << it << ' ';

}

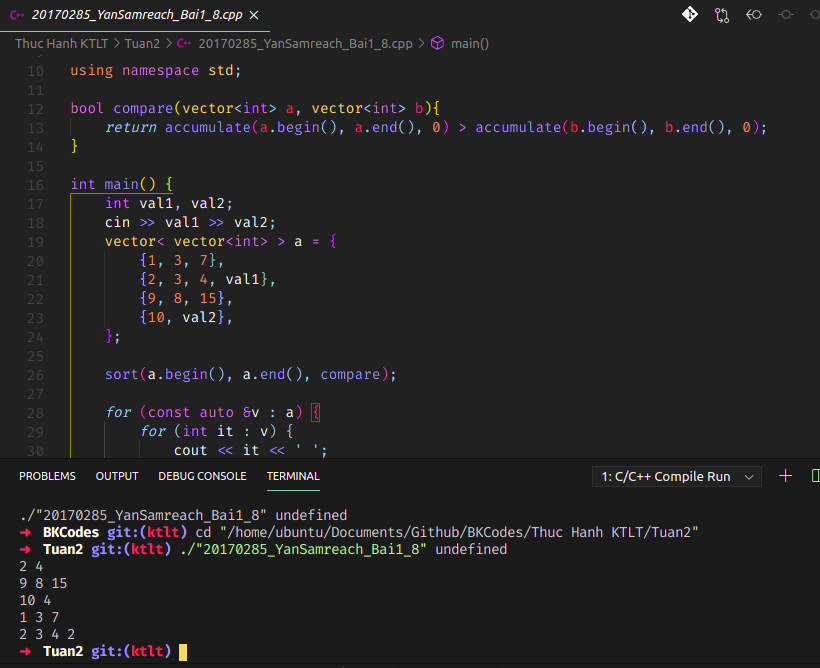
cout << endl;

}

*return* 0;

}

Kết quả:



Hình 8 Bài 1.8 Biểu thức lamda và hàm nặc danh

## Phần 2. Thực hành về tối ưu mã nguồn

### Bài tập 9: Tính hàm sigmoid

*#include* <iostream>

*#include* <math.h>

using namespace std;

double sigmoid\_slow(double x){

*return* 1.0 / (1.0 + exp(-x));

}

int main(){

double xval;

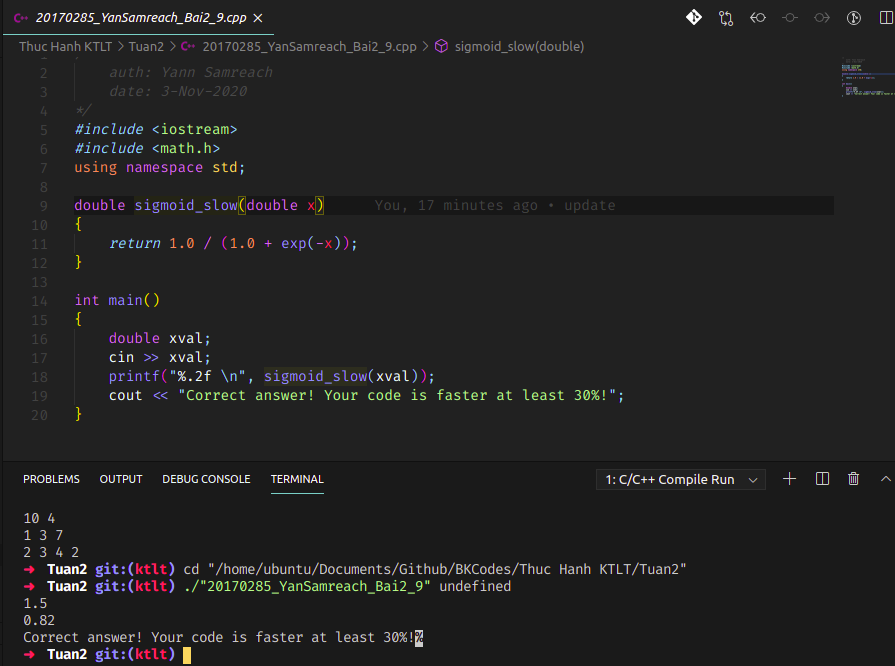
cin >> xval;

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

cout << "Correct answer! Your code is faster at least 30%!";

}

Kết quả:



Hình 9 Bài 1.9 Tính hàm sigmoid

### Bài tập 10: Tính tích hai ma trận vuông

*#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) {

*Matrix c;*

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

*for* (int j = 0; j < N; ++j) {

c.mat[i][j] = 0;

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

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

}

}

}

*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* {

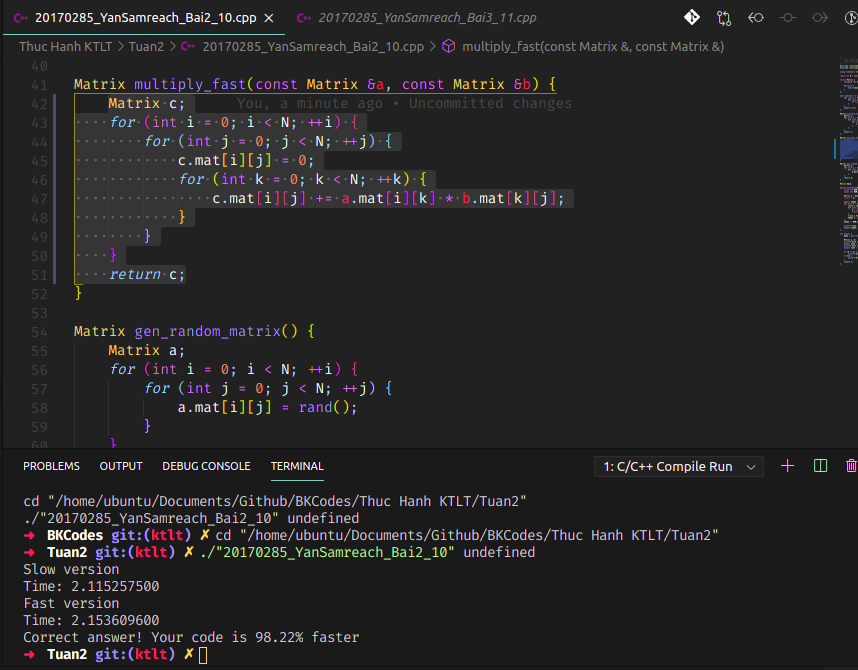
printf("Wrong answer!\n");

}

*return* 0;

}

Kết quả:

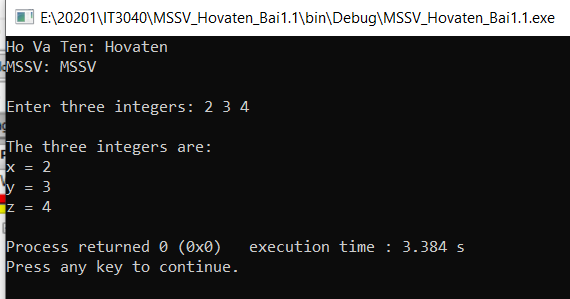


Hình 10 Bài 1.10 Tính tích hai ma trận vuông

## Phần 3. Bài tập về nhà

### **Bài tập 11: Tính tích hai đa thức**

Kết quả:



Hình 11 Bài 1.11 Tính tích hai đa thức

### **Bài tập 12: Map sort**

*#include* <iostream>

*#include* <map>

*#include* <vector>

*#include* <algorithm>

typedef std::pair<int, int> pair;

bool sortByVal(const pair &a, const pair &b) {

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

*return* a.second > b.second;

}

*return* a.first > b.first;

}

int main() {

int a, b;

*// std::map<int, int> map = {*

*// {2, 3}, {4, 8}, {9, 1}, {1, 8}*

*// };*

*// input by keyboard, stop with invalid number*

std::map<int, int> map;

*while* ( std::cin >> a >> b) map.insert(std::pair<int, int>(a,b));

std::vector<pair> vec;

*// copy vector to invert map*

std::copy(map.begin(), map.end(), std::back\_inserter<std::vector<pair>>(vec));

*// sort by value*

std::sort(vec.begin(), vec.end(), sortByVal);

*// print output*

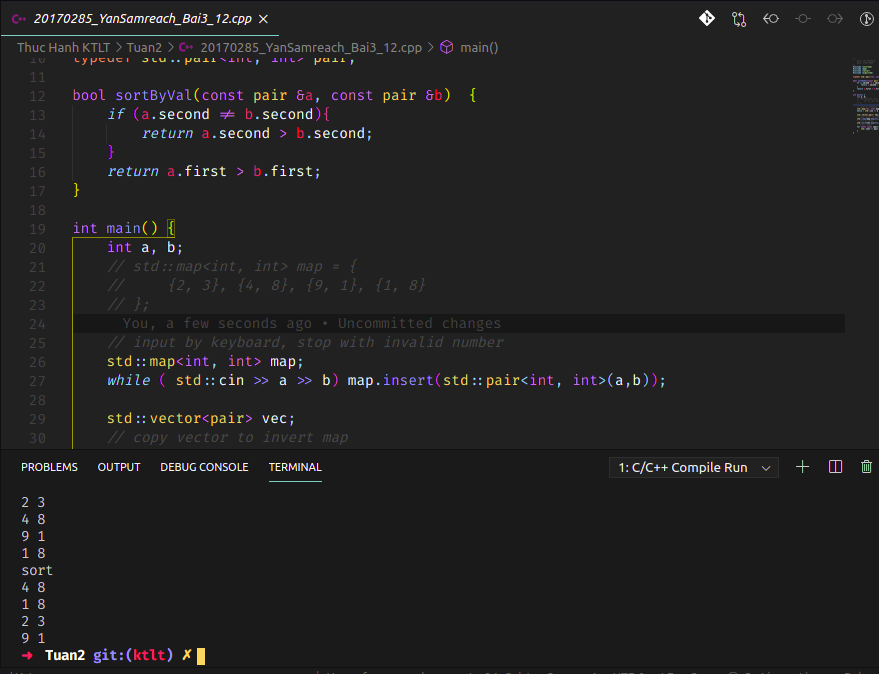
*for* (auto const &pair: vec) {

std::cout << pair.first << " "<< pair.second << std::endl;

}

}

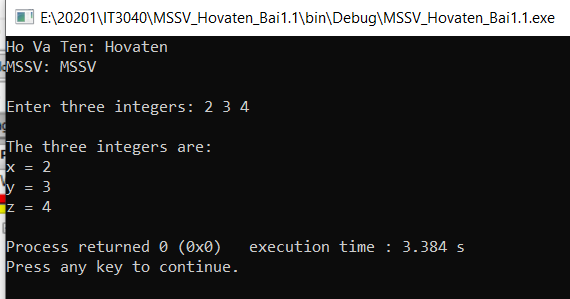
Kết quả:



Hình 12 Bài 1.12 Map sort

### **Bài tập 13: Big integer**

Kết quả:



Hình 13 Bài 1.13 Big integer