Passive Portofolio Manager

Perfectly working soruce code:

Asset\_class header

#pragma once

#include <iostream>

#include <fstream>

#include <vector>

#include <iomanip>

#include <cstdio>

#include <string>

using namespace std;

class asset\_class //it can be: index funds, bond fund, reits

{

public: //need to set it back to private after project is finished (now I need to test functions and print stuff)

string name;

vector<double>price;

vector<double>returns;

vector <double>initialInvestments; // get it from file

vector<double>number\_of\_fund\_units; //calculate it based on purchase price and initial investment

double administrative\_cost;

friend class portofolio;

public:

void get\_name(string s);

void get\_prices(const char\* s);

void get\_admin\_cost();

void get\_investments(const char\* s);//this method reads the initial investments of the investor (for now I will read them from a file)

void return\_calculator(); //it returns the price returns of the asset

double current\_value\_assetClass(); //number of fund units\* current price //it makes it possible for the current market value of portofoliu

void sell\_fund\_unit();

double average\_return();

double value\_of\_assetClass(); //value of investments (price\*fund units)

void print\_stuff(vector<double>v, const char\* s);

void calculate\_number\_of\_units();

double cost\_at\_pf\_value(); //returns the nominal cost of administration APPLIED TO INITIAL INVESTMENTS

double net\_ballance(); //it returns the net profit/loss of the asset class (current market value - invested amount-admin cost)

double total\_investments();

void get\_price\_console(double p);

void get\_investment\_console(double i);

};

Asset\_class.cpp

#include "asset\_class.h"

void asset\_class::get\_price\_console(double p)

{

this->price.push\_back(p);

this->return\_calculator();//after reading the prices calculate the returns

}

void asset\_class::get\_investment\_console(double i)

{

this->initialInvestments.push\_back(i);

this->calculate\_number\_of\_units();//update fund units every time I invest

}

void asset\_class::sell\_fund\_unit()

{

}

void asset\_class::get\_name(string s)

{

this->name = s;

}

double asset\_class::total\_investments()

{

double total = 0;

for (vector<double>::iterator i = initialInvestments.begin(); i != initialInvestments.end(); ++i)

{

total += \*i;

}

return total;

}

double asset\_class::net\_ballance()

{

double current\_mk\_value = current\_value\_assetClass();

double total = this->total\_investments();

double admin\_cost = this->cost\_at\_pf\_value();

return (current\_mk\_value - total - admin\_cost);

}

void asset\_class::get\_admin\_cost()

{

cout << "Admin cost (%): ";

double x;

cin >> x;

x /= 100;

this->administrative\_cost = x;

}

double asset\_class::cost\_at\_pf\_value()

{

return this->total\_investments() \* administrative\_cost;

}

double asset\_class::current\_value\_assetClass() //in order for it to work, the calculate\_number of units method has to be run

{

double nr\_units = 0;

for (vector<double>::iterator i = number\_of\_fund\_units.begin(); i != number\_of\_fund\_units.end(); ++i)

{

nr\_units += \*i;

}

return nr\_units \* price.back();

}

void asset\_class::calculate\_number\_of\_units() //it calculates the number of fund units purchased with every investment

{

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

{

number\_of\_fund\_units.push\_back(initialInvestments[i] / price[i]);

}

}

double asset\_class::value\_of\_assetClass()

{

double value\_of\_pf = 0;

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

{

value\_of\_pf += price[i] \* number\_of\_fund\_units[i];

}

return value\_of\_pf;

}

double asset\_class::average\_return()

{

double sum\_of\_returns = 0;

for (vector<double>::iterator i = returns.begin(); i != returns.end(); ++i)

{

sum\_of\_returns += \*i;

}

return (sum\_of\_returns / returns.size());

}

void asset\_class::return\_calculator()

{

//for(vector<int>::iterator i=price.begin(); i!=price.end();++i)

for (int i = 1; i < price.size(); i++)

{

//returns.push\_back((\*i - (\*i - 1)) / (\*i - 1));

returns.push\_back(((price[i] - price[i - 1]) / (price[i - 1])));

}

}

void asset\_class::print\_stuff(vector<double>v, const char\* s)

{

cout << s << endl;

for (vector<double>::iterator i = v.begin(); i != v.end(); ++i)

{

cout << \*i << " ";

}

cout << endl;

}

void asset\_class::get\_prices(const char\* s)

{

int prices\_count = 0;

ifstream in(s);

if (!in.is\_open())

{

cout << "Could not open file! \n";

return;

}

double x;

while (in >> x)

{

price.push\_back(x);

}

in.close();

// this->return\_calculator();//after reading the prices calculate the returns;

}

void asset\_class::get\_investments(const char\* s)

{

ifstream in(s);

if (!in.is\_open())

{

cout << "Could not open file! \n";

return;

}

float x;

while (in >> x)

{

initialInvestments.push\_back(x);

}

in.close();

// this->calculate\_number\_of\_units();//update fund units every time I invest

}

Portofolio class header

#pragma once

#include "asset\_class.h"

#include <iostream>

#include <fstream>

#include <vector>

#include <iomanip>

#include <cstdio>

#include <string>

class portofolio

{

public:

vector<asset\_class>assets;

double value\_pf;

public:

portofolio()

{

value\_pf=0;

};

void add\_asset(asset\_class a);

double pf\_value(); //it calculates the current value of the portofolio, by adding up all the asset classes

void asset\_allocation(); //it calculates the distribution of different asset classes in the portofolio

void net\_ballance\_per\_asset\_class();

double average\_pf\_return(); //sort out problem

void print\_assets();

};

Portofolio cpp

#include "portofolio.h"

#include "asset\_class.h"

double portofolio::average\_pf\_return()

{

double returns = 0;

//for(int i=0;i<assets.size();i++)

for (vector<asset\_class>::iterator i = assets.begin(); i != assets.end(); ++i)

{

returns += (\*i).average\_return();

}

return ((returns / assets.size()) \* 100);

}

void portofolio::net\_ballance\_per\_asset\_class()

{

cout << "Net ballance of assets: \n";

for (vector<asset\_class>::iterator i = assets.begin(); i != assets.end(); ++i)

{

cout << (\*i).name << ": " << (\*i).net\_ballance() << endl;

}

}

void portofolio::asset\_allocation()

{

vector<double>values;

if (this->assets.empty())

{

cout << "Portofolio is empty!" << endl;

return;

}

cout << "Asset allocation of the portofolio: \n";

for (vector<asset\_class>::iterator i = assets.begin(); i != assets.end(); ++i)

{

cout << (\*i).name << ": " << (((\*i).value\_of\_assetClass()) / this->pf\_value()) \* 100 << " %" << endl;

}

}

double portofolio::pf\_value()

{

double val = 0;

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

{

val += assets[i].value\_of\_assetClass();

}

return val;

}

// try a method for value pf, whichi always returns a valeu

/\* orginial method

void portofolio::pf\_value()

{

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

{

value\_pf += assets[i].value\_of\_assetClass();

}

}

\*/

void portofolio::print\_assets()

{

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

{

cout << assets[i].name << endl;

}

}

void portofolio::add\_asset(asset\_class a)

{

assets.push\_back(a);

this->pf\_value();

}

Main+ menu function

// Passive Portofolio Manager.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include "portofolio.h"

#include "portofolio.h"

#include <string>

#include <cstringt.h>

int get\_command()

{

int cmd;

cin >> cmd;

return cmd;

}

void print\_menu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

//cout << "1. Initialize portofolio" << endl;

cout << "1. Add asset from file" << endl;

cout << "2. Add asset from console" << endl;

cout << "3. Print asset allocation" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void menu()

{

portofolio p;

int x = 100;

while (x != 0)

{

print\_menu();

x = get\_command();

switch (x)

{

case 1: //add from file

{

asset\_class a;

string name;

cout << "Name of asset: ";

cin >> name;

a.get\_name(name);

string file\_investments;

string file\_prices;

cout << "Type in name of file which contains the prices" << endl;

cin >> file\_prices;

a.get\_prices(file\_prices.c\_str()); //works

a.return\_calculator();

cout << "Type in name of file which contains the invested amount " << endl;

cin >> file\_investments;

a.get\_investments(file\_investments.c\_str()); //works

a.calculate\_number\_of\_units();

// a.print\_stuff(a.number\_of\_fund\_units, "fund units"); WORKS

cout << a.name << "Current value: " << a.current\_value\_assetClass() << endl;

p.add\_asset(a);

break;

}

case 2: //add from console

{

asset\_class a;

string name;

cout << "Name of asset: ";

cin >> name;

a.get\_name(name);

double price;

cout << "Price of fund unit: " << endl;

cin >> price;

a.get\_price\_console(price);

double investment;

cout << "Type in the amount you want to invest: ";

cin >> investment;

a.get\_investment\_console(investment);

//a.print\_stuff(a.initialInvestments, "Invested amount read from console");;WORKS

p.add\_asset(a);

p.pf\_value();

break;

}

case 3: //print asset allocation

{

p.asset\_allocation();

break;

}

default:

break;

}

}

}

int main()

{

/\*

asset\_class rotx;

string s="rotx";

//cin >> s;

//rotx.get\_name(s);

cout << rotx.name<<endl;

rotx.get\_prices("input.txt");

//rotx.print\_stuff(rotx.price, "Prices:");

// rotx.return\_calculator();

//rotx.print\_stuff(rotx.returns, "Returns:");

rotx.get\_investments("investments.txt");

rotx.print\_stuff(rotx.initialInvestments, "Initial investments:");

// rotx.calculate\_number\_of\_units();

//rotx.print\_stuff(rotx.number\_of\_fund\_units, "nr fund units");

cout << "total investments: " << rotx.value\_of\_assetClass() << endl;

cout << "CURRENT fund value: " << rotx.current\_value\_assetClass() << endl;

// rotx.get\_admin\_cost();

// cout << "The cost of manageing the portofolio is: " << rotx.cost\_at\_pf\_value() << endl;

//cout << "The current ballance of the portofolio is: " << rotx.net\_ballance();

//testing portofolio class

portofolio myPf;

myPf.add\_asset(rotx);

myPf.asset\_allocation();

myPf.pf\_value();

//cout << myPf.value\_pf << endl; IT CALCULATES THE VALUE OF PF IN THE MAIN CORRECTLY

asset\_class nasdaq;

nasdaq.get\_name("nasdaq");

nasdaq.get\_prices("nasdaq\_price.txt");

nasdaq.get\_investments("nasdaq\_investments.txt");

//nasdaq.print\_stuff(nasdaq.initialInvestments, "nasdaq\_investments");

//nasdaq.print\_stuff(nasdaq.price, "Nasdaq prices");

//nasdaq.calculate\_number\_of\_units();

myPf.add\_asset(nasdaq);

myPf.print\_assets();

myPf.pf\_value();

cout<<"The value of the portofolio is: " << myPf.value\_pf;

myPf.asset\_allocation();

myPf.net\_ballance\_per\_asset\_class();

//nasdaq.return\_calculator();

// nasdaq.print\_stuff(nasdaq.returns, "returns nasdaq");

cout << "Average Return of the portofolio " << myPf.average\_pf\_return()<<endl;

\*/

menu();

}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu

// Debug program: F5 or Debug > Start Debugging menu

// Tips for Getting Started:

// 1. Use the Solution Explorer window to add/manage files

// 2. Use the Team Explorer window to connect to source control

// 3. Use the Output window to see build output and other messages

// 4. Use the Error List window to view errors

// 5. Go to Project > Add New Item to create new code files, or Project > Add Existing Item to add existing code files to the project

// 6. In the future, to open this project again, go to File > Open > Project and select the .sln file