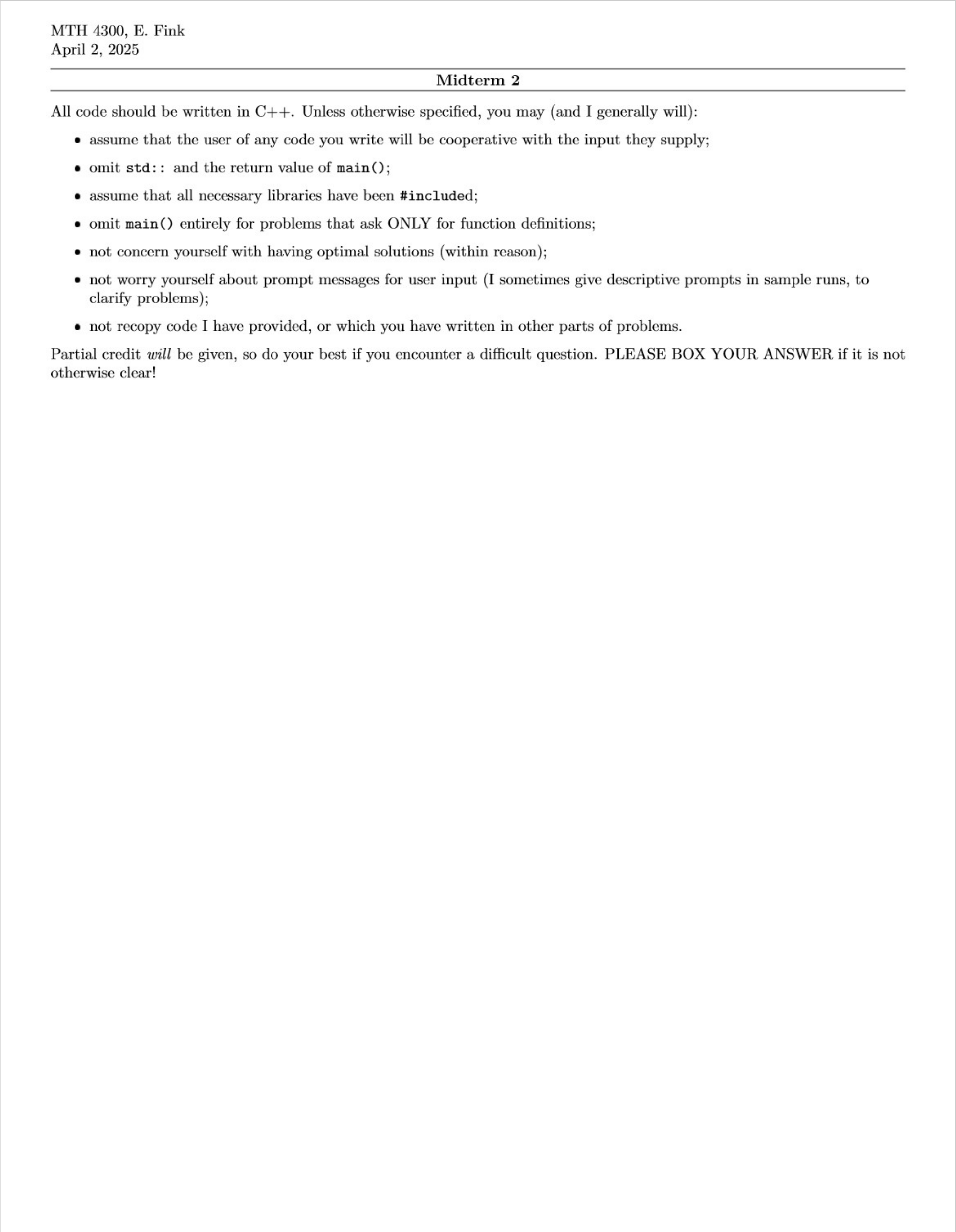
Spring 24 Sample Midterm 2

Wednesday, April 2, 2025 11:13 PM

Spring 24
Sample M...



MTH 4300, E. Fink
April 2, 2025

Midterm 2

All code should be written in C++. Unless otherwise specified, you may (and I generally will):

- assume that the user of any code you write will be cooperative with the input they supply;
- omit `std::` and the return value of `main()`;
- assume that all necessary libraries have been `#include`d;
- omit `main()` entirely for problems that ask ONLY for function definitions;
- not concern yourself with having optimal solutions (within reason);
- not worry yourself about prompt messages for user input (I sometimes give descriptive prompts in sample runs, to clarify problems);
- not recopy code I have provided, or which you have written in other parts of problems.

Partial credit *will* be given, so do your best if you encounter a difficult question. PLEASE BOX YOUR ANSWER if it is not otherwise clear!

1. I have a file named **shopping.txt**, which contains several sentences, which looks like:

```
I buy 5 computers.
I buy 13 books about programming.
I buy 25 used keyboards.
```

The file could have several lines, but each sentence will be on a line by itself, and start with "I buy" followed by a number, and an item.

Write a program which opens this file, and prints out the total number of items I buy. (For example, with the file shown, the number printed would be 43, since 5 + 13 + 25 = 43.) I suggest using stringstreams.

For full credit, your program should work for any file named **shopping.txt**, which contains some number of sentences which begin with "I buy" followed by an int.

```
int main(){
    ofstream f("shopping.txt");
    int total=0;
    string line;
    while (getline(f, line)){
        stringstream s;
        s<<line;
        string x;                "I"       "buy"
        int y;
        s>>x  >>x  >>y;
        total +=y;
    }
    cout<<y;
```

2. You are interested in sports analytics. You want to write a program to analyze how teams perform through a season.

You want to create a class called `Team`. Each object is meant to represent a football team. Each `Team` object should have the following **private member variables**:

- `string name`, the **name of the team**;
- `string opps[16]`, an array of strings representing the opponents in the 16 games of the regular season, which should be initialized with each entry set to `"---"`;
- `int score[16]` and `int opp_score[16]`, representing the scores of each team in question and their opponent in each of the games.

The class should also support the following **methods**:

- a constructor which receives one `string` parameter which sets `opps` as described above and each entry of `score` and `opp_score` to `-1` (signifying the game has yet to be played).
- `void add_game(int game_no, string opp, int s_us, int s_them)`. This method should check that `game_no` is an appropriate value (`0` to `15`). If it is, it should update all the attributes, by setting index `game_no` in all three arrays to equal the corresponding argument.
- `double win_percent()`, which returns the percentage of games that have been played so far that our team has won (if no games have been played yet, then `0` should be returned). Games that have been played can be identified by checking if the opponent is `"---"`.

a. Write the declaration for the class `Team`, and the definitions for all methods. Mark methods as *const* as appropriate.

b. Write code in `main()` which declares a `Team` object for a team called the `Tigers`. Data for games 0 and 1 should be supplied: the opponents are the `Bulldogs` and `Gators`; the respective scores are `3-28` and `10-0` (the Tigers' score is listed first). Then, print out Tigers' win percentage (which should be 0.5 or 50% – I don't care about the format).

```
class Team {
Private:
        String name;
        String opps[16];
        int score[16], opp_Score [16];
public:
      Team(string);
      void  add_game(int, string, int, int);
      double  win_perent () const;
};

 Team:: Team (string s) {
      name=s;
      for(int i=0;  i<15;  ++i) {
            opps[i] = "---";


            score [i] = -1;
            opp_score[i] = -1;
        }
  }
```

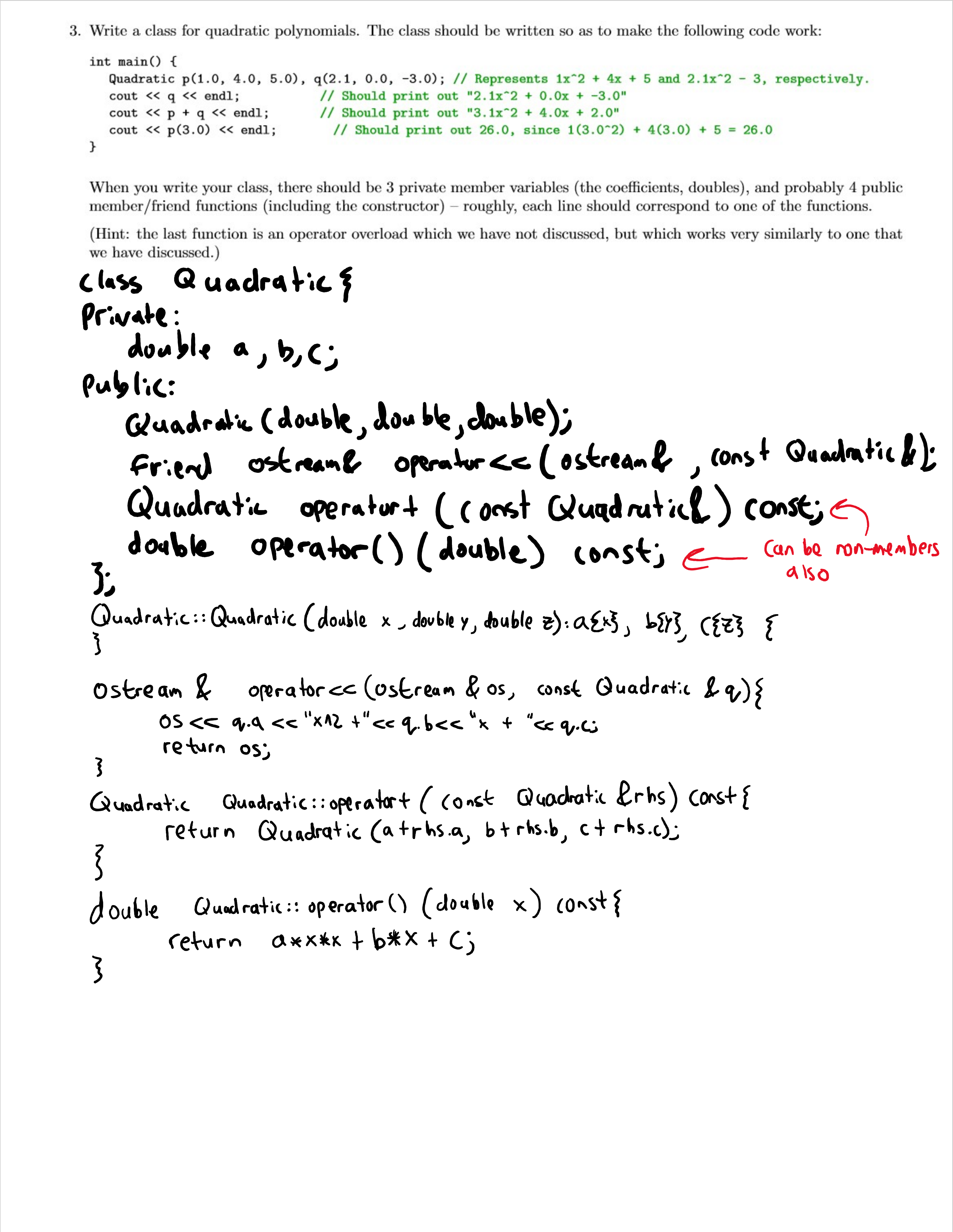
```
void Team::add_game (int g, string o, int us, int them){
     if (0<=g  && g<=15){
          opps[g] = o;
          score [g] = us;
          opp_score [g] = them;
     }

double Team:: win_percent ( ) const {
     int  games=0,  wins=0;
     for (int  i=0;  i<16;  ++i){
          if (opps[i]  != "---"){
               ++ games;
               if ( score[i] > opp_score[i]){
                    ++wins;
               }
          }
     }
     if ( games!=0){
          return   wins*100.0/games;
```

Puts into percent form (unimportant)
AND AVOIDS int DIVISION!

```
     }
     return 0;
}

int main(){
     Team  x("Tigers");
     X. add_game(0, "Bulldogs", 3, 28);
     X.add_game(1, "Gators", 10,0);
     cout << X. win_percent()<< endl;
}
```



3. Write a class for quadratic polynomials. The class should be written so as to make the following code work:

```
int main() {
   Quadratic p(1.0, 4.0, 5.0), q(2.1, 0.0, -3.0); // Represents 1x^2 + 4x + 5 and 2.1x^2 - 3, respectively.
   cout << q << endl;           // Should print out "2.1x^2 + 0.0x + -3.0"
   cout << p + q << endl;       // Should print out "3.1x^2 + 4.0x + 2.0"
   cout << p(3.0) << endl;        // Should print out 26.0, since 1(3.0^2) + 4(3.0) + 5 = 26.0
}
```

When you write your class, there should be 3 private member variables (the coefficients, doubles), and probably 4 public member/friend functions (including the constructor) – roughly, each line should correspond to one of the functions.

(Hint: the last function is an operator overload which we have not discussed, but which works very similarly to one that we have discussed.)

```
class Quadratic{
Private:
    double a, b, c;
Public:
    Quadratic(double, double, double);
    friend ostream& operator<<(ostream& , const Quadratic&);
    Quadratic operator+ (const Quadratic&) const;
    double operator() (double) const;
};
```

Can be non-members also

```
Quadratic::Quadratic(double x, double y, double z): a{x}, b{y}, c{z} {
}

ostream & operator<<(ostream & os, const Quadratic & q){
      os << q.a << "x^2 +"<< q.b<< "x + "<< q.c;
      return os;
}

Quadratic Quadratic::operator+ (const Quadratic &rhs) const{
      return Quadratic(a+rhs.a, b+rhs.b, c+rhs.c);
}

double Quadratic::operator() (double x) const{
      return a*x*x + b*x + c;
}
```

4. Consider the following code, which doesn't do anything interesting, but will cause a memory leak if used:

```
class Leaky {
private:
    int *ptr;
    string identity;
public:
    Leaky() {
        ptr = new int[100];
        identity = "???";
        cout << identity << endl;
    }

    Leaky(const string& x) {
        ptr = new int[100];
        identity = x;
        cout << identity << endl;
    }
    Leaky(const Leaky& rhs) {
        ptr = rhs.ptr;
        identity = rhs.identity + "copy";
        cout << identity << endl;
    }

    ~Leaky() {
        cout << identity << "BYEBYEBYE" << endl;
    }
};

int main(){
    cout << "111" << endl;
    Leaky a;
    Leaky b("Fred");
    cout << "222" << endl;
    b = a; → NO CONSTRUCTOR CALL
    cout << "333" << endl;
    Leaky c = b;
    cout << "444" << endl;
}
```

$\boxed{a}$

```
111
???
Fred
222
333
???copy
444
???copy BYEBYEBYE
??? BYEBYEBYE
??? BYEBYEBYE
```

Destructor calls (you can write these in any order for full credit)

Sets b.ptr to be a.ptr, and b.identity to be ???

a. What would print out when the code above is run? (Note: the last three lines to print should do so in a particular order, but I did not discuss this and so won't deduct points if the last 3 lines are permuted.)

b. This class will cause memory leaks because heap-allocated memory is not deallocated when objects go out of scope. Write one extra line in one of the member functions that "solves" this problem, and be sure to be clear about which member function you're adding it to.

c. The reason that I put "solves" in quotation mark is because the one extra line, on its own, will stop the memory leak, but it will cause other problems – for example, if the code above is run with just the change of part b, there will be probably be a runtime error. Explain why an error occurs with this code, as clearly as you can.

b. Add to ~Leaky():

delete[] ptr;

c. Class disobeys the "Rule of 3": No operator=()

AND

Copy constructor performs shallow copy.

So a, b, c all end up pointing to same array.

Therefore, delete[] will get called 3 times on same heap-allocated array!