CS 2124 Exam #1

Barry Zhang

TOTAL POINTS

90.5 / 103

QUESTION 1

1 Q1 (Extra Credit) 3 / 3

✓ - 0 pts Correct

QUESTION 2

2 Q2 5 / 5

✓ - 0 pts Correct

QUESTION 3

3 Q3 5 / 5

✓ - 0 pts Correct

QUESTION 4

4 Q4 5 / 5

✓ - 0 pts Correct

QUESTION 5

5 Q5 8 / 10

Function signature

✓ - 1 pts Used initialization list

Functionality

✓ - 1 pts Fails to copy size

QUESTION 6

6 Q6 5 / 5

✓ - 0 pts Correct

QUESTION 7

7 Q7 5 / 5

✓ - 0 pts Correct

QUESTION 8

8 Q8 0 / 5

✓ - 5 pts Incorrect

QUESTION 9

9 Q9 15 / 15

✓ - 0 pts Correct

fillData

✓ - 0 pts Correct

filter

✓ - 0 pts Correct

QUESTION 10

Q10 20 pts

10.1 Q10A 7 / 7

✓ - 0 pts Correct

10.2 Q10B 6 / 6

Signature

✓ - 0 pts Correct

Loop structure

✓ - 0 pts Correct

Calculation

✓ - 0 pts Calculation using integer division

10.3 Q11C 7 / 7

✓ - 0 pts Correct

Signature

✓ - 0 pts Correct

Loop to free

✓ - 0 pts Correct

QUESTION 11

11 Q12 19.5 / 25

Constructor

✓ - 0 pts Correct

Output operator

✓ **- 0.5 pts** "rhs."

worksWith

✓ **- 5 pts** **Parameter passed by value or constant reference**

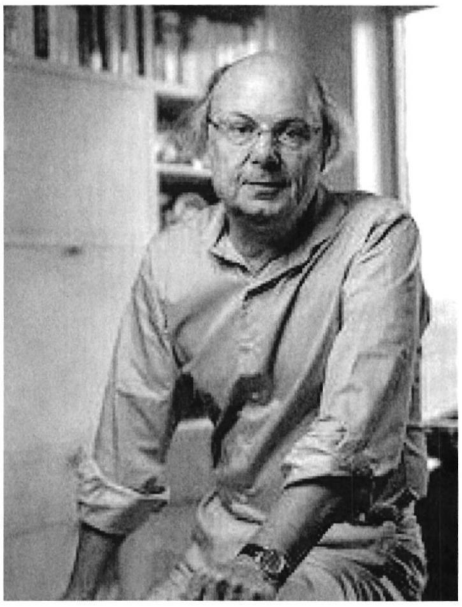
Name Barry Zhang Net ID: bz1148

CS-UY 2124 - Object Oriented Programming
MID-TERM EXAM #1 - October 26th, 2021

- **Do not open this test booklet until you are instructed to do so**
- Duration: 1 hour, 15 minutes
- Do not separate any pages. Do not pull the test apart from the staple.
- Ensure your name and Net ID is printed at the top of every page.
- This is a closed book exam, no calculators, computers, or phones are allowed
- Anyone found cheating on this exam will receive a zero for the exam
- Anyone who is found writing after time has been called will receive a zero for this exam
- If you have a question please ask the proctor of the exam.
- Note that we have omitted any `#includes` or `using namespace std;` statements in all questions in order to save space and to save your time thinking about them. You may assume that all such statements that are needed are present. And you don't have to write them either!!!
- You also do not need to write any comments in any of your code.
- Please read all questions carefully! They may look familiar and yet be completely different.
- Answering the short-answer questions, in particular, requires that you read and understand the programs shown. You need to read them carefully if you are going to understand them.
- If a question asks you to write a class or a function and provides you with test code, **be sure your class / function works with that test code**. If the question provides you with sample output, then your answer should match that output.
- Print your name and Net ID on the top of **EACH** page.

Name _____ Net ID: bz1148

1. **EXTRA CREDIT:** Who created C++? Circle the correct answer (including the item letter and the name).



   A. Bill Gates
   B. Sergey Brin
   C. Guido van Rossum
   D. Ada Lovelace
   E. Alan Turing
   F. James Gosling
   G. Bjarne Stroustrup
   H. Dennis Ritchie
   I. Claude Shannon
   J. None of the above

2. **(5 pts.)** Given the following code, what value would be output when the **`calc_volume()`** method is invoked on the object **`cyl`**? Circle the correct answer (including the item letter).

```
const double PI = 3.14;

class Cylinder {
public:
    double calc_volume() const {
        return PI * radius * radius * height;
    }

private:
    double radius;
    double height;
};

int main() {
    Cylinder cyl;

    cout << "Cylinder volume: " << cyl.calc_volume() << endl;
}
```

   A. 1
   B. 0
   C. undefined
   D. 3.14
   E. none of the above

Name _____ Net ID: bz1148

3. **(5 pts.)** Consider the **`Cylinder`** class defined above. If a pointer to a **`Cylinder`** named **`c_ptr`** exists in a program, which of the following expressions invokes the **`calc_volume()`** method on the **`Cylinder`** "pointed at" by **`c_ptr`**? Circle the correct answer (including the item letter).

A. `c_ptr.calc_volume()`

B. `(*c_ptr.calc_volume())`

C. `*(c_ptr.calc_volume())`

D. `c_ptr->calc_volume()`

E. `c_ptr>-calc_volume()`

F. `c_ptr<-calc_volume()`

G. Any of the above

H. None of the above

4. **(5 pts.)** Consider the existence of a class named **`Point`**. Two **`Point`** objects have been instantiated by the statements below:

```
Point pt1(3.2, 4.6);
Point pt2(9.1, -4.2);
```

A third **`Point`** object is instantiated by the statement below.

```
Point pt3 = pt1;
```

The statement above produces which of the following function calls? Circle the correct answer (including the item letter).

A. `pt1.assignment=(pt3)`

B. `pt1.operator=(pt3)`

C. `pt3.assignment=(pt1)`

D. `pt3.operator=(pt1)`

E. `the Point copy constructor`

F. `Point(pt1.x, pt.y)`

G. `Point(pt3.x, pt3.y)`

H. `None of the above`

Name _____ Net ID: 321148

5. **(10 pts.)** Given the following definition of the **`CellPhone`** class,

```
class CellPhone {
public:
  CellPhone (double size = 0)
      : size(size) {bt = new Battery; }
  ~CellPhone() { delete bt; }

  ... // copy constructor definition

private:
  double size;
  Battery* bt;
};
```

Assume that a definition of the **`Battery`** class exists and includes a copy constructor. Implement the assignment operator for the **`CellPhone`** class such that it can be invoked by the assignment statement below

```
CellPhone phone1, phone2;
... // things happen with the cell phones
phone1 = phone2;
```

```
CellPhone& (const CellPhone& rhs) : size(rhs.size) {
    if (this != &rhs) {
        delete bt;
        bt = new Battery(*rhs.bt);
    }
    return *this;
}
```

Name _____ Net ID: bz1148

6. **(5 pts.)** Consider a program defining a class named **`Pirate`**. An overloaded output operator function exists that accepts a **`Pirate`** instance as its second parameter. The program creates a vector of **`Pirate`** objects named **`pirates`**:

```
vector<Pirate> pirates;
```

The vector is populated with **`Pirate`** objects. Write a ranged for loop that outputs each Pirate on a separate line:

```
for (const Pirate& aPirate : pirates) {
    cout << aPirate << endl;
}
```

Name _____ Net ID: bz1148

7. **(5 pts.)** Given the program below, what would be the result of compiling and executing the code.

```
double* do_things(const double& num) {
  double* const dbl_ptr = &num;       // line 1
  cout << *dbl_ptr << ' ';            // line 2
  dbl_ptr = new double(num + 1);      // line 3
  return dbl_ptr;                     // line 4
}

int main() {
  double val = 3.4;                          // line 5
  double* result_ptr = do_things(val);       // line 6
  cout << *result_ptr << ' ';                // line 7
  delete result_ptr;                         // line 8
}
```

A. Compilation error at line 1

B. Compilation error at line 2

C. Compilation error at line 3

D. Compilation error at line 4

E. Program will output `3.4` followed by a compilation error

F. Program will output: `3.4 4.4`

G. Compilation error at line 5

H. The program will output: `4.4 3.4`

I. Compilation error at line 6

J. Compilation error at line 7

K. Compilation error at line 8

L. Program will output: `3.4`

M. None of the above

Name Barry Zhang Net ID: bz1148

8. **(5 pts.)** What would be observed after compiling and executing the program below?

```
vector<int>* add_elems(vector<int> vec) {
    vec.push_back(2);
    vec.push_back(1);
    vec.push_back(2);
    vec.push_back(4);

    return &vec;
}

int main() {
    vector<int> int_vc;
    vector<int>* ptr = add_elems(int_vc);

    for (size_t i = 0; i < ptr->size(); i++) {
        cout << (*ptr)[i] << ' ';
  }
}
```

A. Compilation error

B. Program crashes but compiles successfully

C. Memory addresses are output

D. Undefined behavior

E. Program will output: `2 1 2 4`

F. Program will output: `2`

G. Program will output: `1`

H. Program will output: `4`

I. None of the above

Name _____ Net ID: _____

9. **(15 pts.)** Write two functions:
   - one will read a tab-separated (`\t`; tabs are whitespace) input data file that contains data about pizzas that our pizza store sells and which fills a vector that stores the data structures that hold each row of data, and
   - one that will display those vector items which match a filter requirement.

The Input File
Each line of the file is formatted as shown below. Each line of the input file should be represented as a **`Pizza`** struct. (See the example data file.)

The struct **`Pizza`** is defined as:

```
struct Pizza {
    int size;
    string topping;
    double price;
};
```

fillData Function
Implement the **`fillData`** function. Your implementation should accept the input stream and a vector for the data. The function should populate the vector and return nothing.

filter function
Implement the **`filter`** function. Your implementation should accept the data vector and a double value. Print any vector items which have a price that is **greater than or equal** to the double parameter. Output any format you wish, but include all values in the output of each item.

Example of calling the `fillData` and `filter` functions from `main`
As an example, the functions could be called as:

```
int main() {
   vector<Pizza> inventory;
   ifstream ifs("input.txt");
   if (!ifs) {
      cerr << "failed to open input.txt";
      exit(1);
   }
   fillData(ifs, inventory);
   filter(inventory, 8.00);
}
```

Example of the input file

```
8     plain          8.75
12    sausage        13.50
8     broccoli       7.75
10    onion          7.75
14    xtra_cheese    13.00
8     ham            8.00
```

Example output

```
8     plain          8.75
12    sausage        13.5
14    xtra_cheese    13.0
8     ham            8.00
10    sausage        12.3
8     pepperoni      8.6
```

Name Barry Zhang Net ID: bz1148

```
6     pineapple      5.50
10    sausage        12.30
6     pepperoni      5.50
8     pepperoni      8.60
```

(Answer for Question 9 goes here.)

```
void fillData ( ifstream &  ifs , vector<Pizza>& inventory ) {

    int anInt ;
    string aString;
    double aDouble;

    while ( ifs >> anInt ){
        ifs >> aString >> aDouble ;
        Pizza aPizza {anInt, aString, aDouble };
        inventory.push_back ( aPizza);
    }
}

void filter( const vector<Pizza>& inventory , double thePrice)
    for ( const Pizza& aPizza : inventory ){
        if(( aPizza.price == thePrice) || (aPizza.price > thePrice)){
            cout << aPizza.size\t << aPizza.topping\t ;
            cout << aPizza.price << endl;
        }
    }
}
```

Name Barry Zhang Net ID: bz1148

10. **(20 pts.)** Given the following definition of a **`Pixel`** struct, complete the 3 sub-questions below:

```
struct Pixel {
  int red;
  int blue;
  int green;
};
```

Part A

Define a function that accepts an **`ifstream`** and vector of **`Pixel`** **pointers**. The input file contains 3 integers per line with each integer corresponding to the red, blue, and green RGB colors (in that order) for a pixel. An example file could contain the following lines:

File:

```
0 23 56
9 245 3
255 67 38
100 0 98
```

The function fills the vector with pointers to Pixel objects, one for each line in the file.
[Put you answer to Part A here]

```
void ( ifstream& file , vector<Pixel*>& Pixels ) {
    int red, blue, green;
    while ( file >> red ) {
        file >> blue >> green;
        Pixel* aPixel = new Pixel { red, blue, green };
        Pixels. push_back ( aPixel ) ;
    }
}
```

Name Barry Zhang Net ID: bz1148

Part B

Define a function that accepts a vector of **`Pixel` pointers**. For every **`Pixel`** with a pointer stored in the vector, calculate the grayscale value for the pixel and output it to a separate line of standard output. The grayscale value for a pixel is calculated as the average of the **`Pixel`**'s red, blue, and green values.
[Put you answer to Part B here]

```
void grayScale( const vector<Pixel*>& Pixels ) {
    int gray;
    for ( size_t i=0 ; i< Pixels.size() ; ++i ) {
        gray = (Pixels[i]->red + Pixels[i]->blue + Pixels[i]->green)/3;
        cout << gray << endl;
    }
}
```

Part C

Define a function that frees the memory allocated on the heap for each item in a vector of **`Pixel`** pointers. Do not leave any dangling pointers.
[Put you answer to Part C here]

```
void free( vector<Pixel*>&Pixels ) {
    for ( size_t i=0 ; i< Pixels.size() ; ++i ) {
        delete Pixels[i];
        Pixels[i] = nullpter;
    }
}
```

Name _____ Net ID: _____

12. **(25 pts.)** A `Tool` is defined as follows

    - it contains a name (string),
    - a number of volts (int), and
    - a collection of `Tool`s that can be used with this `Tool` (supports the use of this `Tool`). This collection must be a collection of **pointers** to `Tool`s. Note that this is a one-way relationship not a reciprocal relationship. For example, a hammer need not support another `Tool`, but perhaps a belt sander would support the use of a screwdriver and a jigsaw.

    Provide the definition of the `Tool` class. This is the ONLY class you need to write.

    Implement the following functions

    - an appropriate constructor for the `Tool` class (see the test code below)
    - overload the output operator for a `Tool` providing the information about the `Tool`. This must include the name, voltage and the names of the tools that can be used with this `Tool`, if applicable. (See example output below.)
    - a method `works_with` which adds a `Tool` to the vector of `Tool`s that supports the use of this `Tool`.

    Enforce the following rules

    - a `Tool` can not be used with itself,
    - a `Tool` cannot be added more than once to the "supports" collection for another `Tool` (but can work with more than one `Tool`). That is, a screwdriver can support both a jigsaw and a hammer, but can not appear multiple times in either the hammer or jigsaw's "supports" collection
    - the `works_with` method should not fail silently.

    Note

    - This problem does not involve copy control or the heap.
    - `Tool` names are not unique! Do not compare two `Tool` objects by their name!

(continued next page)

Name _____ Net ID: _____

Sample test code

You should consider the following code to test your implementation:

```
Tool hammer("Hammer", 0);
Tool screwdriver("Screwdriver", 10);
Tool jigsaw("Jigsaw", 175);
Tool belt_sander("Belt Sander", 100);

vector<Tool*> my_equipment{&hammer, &screwdriver, &jigsaw,
                           &belt_sander};

hammer.works_with(hammer);              // returns false
screwdriver.works_with(hammer);         // returns true
jigsaw.works_with(belt_sander);         // returns true
jigsaw.works_with(hammer);              // returns true

for (const Tool* t: my_equipment) {
    cout << *t << endl;
}
```

Sample output

The following output was produced from executing our sample test code on our solution:

```
Hammer: (0)
Screwdriver: (10) works with: Hammer
Jigsaw: (175) works with: Belt Sander      Hammer
Belt Sander: (100)
```

(Provide your solution on next page)

Name Barry Zhang Net ID: bz1148

(Answer for Question 12 goes here.)

```
class Tool {

    friend ostream& operator<<(ostream& os, const Tool& rhs);

string name;
int volts;
vector<Tool*> Tools

public:
    Tool(const string& name, int volts): name(name), volts(volts) { }
    bool inCollect(const Tool& aTool){
        for (size_t i=0; i<Tools.size(); ++i){
            if (&aTool == Tools[i]){
                return true;
            }
        }
        return false
    }
    bool works_with(const Tool& aTool) {
        bool tF = inCollect(aTool);
        if (&aTool == this) || (inCollect) {
            return false,
        }
        else {
            Tools.push_back(&aTool);
            return true;
        }

    }
};
```

// next page →

Name Barry Zhang Net ID: bz1148

(Additional space for Question 12.)

```
ostream& operator<<(ostream& os, const Tool& rhs) {
    os << name << " : (" << volts << ") ";
    if (rhs.Tools.size() != 0) {
        os << "works with: ";
        for (size_t i=0; i< rhs.Tools.size(); ++i) {
            os << rhs.Tools[i] -> name << "\t";
        }
    }
    return os;
}
```