

EE 200 Exam 2

Tufts University

6 November 2019

Name: _____

Question	Points
Matrix2D	14
Heap memory	7
References and copies	11
Inheritance, polymorphism, and templates	8
What gets printed?	11
Total	51
Bonus	3

Instructions:

1. This examination contains 10 pages, including this page.
2. You have **seventy-five (75) minutes** to complete the examination.
3. Write your answers in this booklet. We scan this into Gradescope, so scratch work on other pieces of paper will not be scanned or counted for credit.
4. You can assume the relevant headers and namespaces have been included or imported in any code snippets that need them.
5. Remember that `uint8_t`, `int32_t`, `uint64_t`, and so forth are typedefs that correspond to unsigned and signed integers of the given widths. These are used so that you don't have to worry about how big an `int` is on your platform.

Question 1: Matrix2D class

The Matrix2D class represents a 2-dimensional matrix of double, which can be used to represent matrices and vectors for linear algebra.

The partial class declaration is below:

```
class Matrix2D {  
private:  
    double* data;  
    int rows; // height, aka number of rows  
    int cols; // width, aka number of columns  
  
public:  
    // FILL IN: constructor declaration  
  
    // FILL IN: copy constructor declaration  
  
    // FILL IN: destructor declaration  
  
    // FILL IN: overloaded () operator  
  
};
```

- (a) [2 pts] In the space provided above in the class declaration, write the declaration for a constructor which takes two dimensions (rows and columns). All array elements should be initialized to zero.
- (b) [2 pts] Write the implementation of this constructor below:

- (c) [2 pts] In the space provided above in the class declaration, write the declaration for the copy constructor.
- (d) [2 pts] Write the implementation for the copy constructor below. The copy constructor should make a deep copy of the object.

- (e) [2 pts] In the space provided above in the class declaration, write the declaration for the destructor.
- (f) [2 pts] Write the implementation of the destructor below. Needless to say, the destructor should clean up so that there are no memory leaks.

(g) [3 pts] In the space provided above in the class declaration, write the declaration for an overloaded `()` operator for this class, which takes two integer arguments (row and column) and returns a lvalue reference to the element at the corresponding position. This will enable a user to write code such as `matrix(2, 0) = 5;`.

(h) [3 pts] Write the implementation for the overloaded `()` operator.

Question 2: Heap memory

(a) [2 pts] What is the difference between `free()` and `delete`?

(b) [3 pts] What is the “rule of three”, and why is it a rule?

- (c) [2 pts] Memory allocated by `malloc` is freed when (select all that apply):
- (a) The pointer which points to it goes out of scope
 - (b) The `free` function is called with the pointer which points to it
 - (c) There are no more valid references to the memory
 - (d) The thread on which the memory was allocated is joined back to the main thread
 - (e) The program exits

Question 3: References and copies

- (a) [3 pts] Why did C++ add references in addition to pointers? In other words, what does a reference do that a pointer cannot?

- (b) [4 pts] In the code below, how many `std::string` objects are created (i.e., how many times is one of the string constructors called)?

```
class Book
{
public:
    Book(const string author, const string title);
    void setAuthor(const string author);

private:
    string mAuthor;
    string mTitle;
    int mPublicationYear;
};

Book::Book(const string author, const string title)
{
    mAuthor = author;
    mTitle = title;
}

Book::setAuthor(const string author)
{
    mAuthor = author;
}

int main(int argc, char* argv[])
{
    Book b("Andrew Hilton", "All of Programming");
    b.setAuthor("Anne Bracy");
}
```

- (c) [4 pts] Update the code above to eliminate unnecessary copies while achieving the same function. Cross out lines or write in the margins as necessary.

Question 4: Inheritance, polymorphism, and templates

(a) [2 pts] Indicate whether inheritance would be appropriate for the following objects:

- Directory : File
- InputDevice : Keyboard
- List : SortedList
- MainWindow : PushButton

(b) [3 pts] What is the difference between static and dynamic dispatch?

(c) [3 pts] Why is templated code usually placed in a header file?

Question 5: What gets printed?

Given the following declarations:

```
class Animal {
public:
    const char* noise() { return "animal noise"; }
};

class Cow : public Animal {
public:
    const char* noise() { return "MOOO!"; }
};

class UnknownError : public exception {
public:
    virtual const char* what() const throw()
    { return "There's a sound that no one knows."; }
};

class Fox : public Animal {
public:
    const char* noise() { throw UnknownError(); }
};
```

- (a) [4pts] What is printed when the following code executes? If there is a compiler error or the result is undetermined, say so.

```
using namespace std;

int main(int argc, char* argv[])
{
    Animal* bessie = new Cow();
    Cow* buttercup = new Cow();
    cout << "bessie says " << bessie->noise() << endl;
    cout << "buttercup says " << buttercup->noise() << endl;
}
```

- (b) [4 pts] What is printed when the following code executes? If there is a compiler error or the result is undetermined, say so.

```
int main(int argc, char* argv[])
{
    try {
        Fox* tails = new Fox();
        cout << "----" << endl;
        cout << "Fox says " << tails->noise() << endl;
        cout << "----" << endl;
    }
    catch (const exception& e) {
        cout << e.what() << endl;
    }
    cout << "goodbye" << endl;
}
```

- (c) [3 pts] What is printed when the following code executes? If there is a compiler error or the result is undefined, say so.

```
void* run(void* name)
{
    cout << (char*)name << endl;
}

int main(int argc, char* argv[])
{
    pthread_t hare;
    pthread_t tortoise;

    int ht = pthread_create(&hare, NULL, run, (void *)"Hare!");
    int tt = pthread_create(&tortoise, NULL, run, (void *)"Tortoise.");

    pthread_exit(NULL); // Quit main but let threads finish running
}
```

Question 6: Bonus

In an online discussion of “little-known features of C and C++”, someone shared the “goes-to operator”, demonstrated below.

```
for(unsigned int i = 5; i --> 0;){ // i goes to zero
    printf("i: %d\n", i);
}
```

- (a) [2 pts] What does this code print out, and why? (In case you’re wondering: it is legal C, and compiles just fine. You have seen it before, although it might look a little different.)

```
for(unsigned int i = 0; i --> 5;){ // i goes to 5?
    printf("i: %d\n", i);
}
```

- (b) [1 pt] What does this code print out, and why?