CS 7B - Spring 2018 - Final Exam

Write your responses to following questions on this paper, or attach extra, as needed. Use complete sentences where appropriate and write out code using proper style and syntax.

1. Suppose that the variable `x` is stored in the memory address `0x28ff04`. What will the following C++ code print (assume it doesn't crash)?

```
#include <iostream>

int main() {
    int x = 17;
    int* y = &x;
    int& xref = x;
    std::cout << ++x << std::endl;
    std::cout << *y << std::endl;
    std::cout << ++xref << std::endl;
    std::cout << &xref << std::endl;
    std::cout << ++y << std::endl;
    std::cout << *y << std::endl;
}
```

Why might it crash?

ANS:

```
18
18
19
0x28ff04
0x28ff08
2686724 <-some derefenced random memory location
```

It might crash because it's reading from unallocated memory and reading from unallocated memory is "undefined behavior"–there's no telling what might happen!

2. Describe the output you might expect from the following program.

```
#include <iostream>
int main() {
    int x = 65;
    int* y = &x;
    for(int i = 0; i < 100; ++i) {
        std::cout << (char)*(y++);
    }
}
```

Why is this program more likely to crash than the program in question # 1?

ANS: If the program doesn't crash, it will output the 100 bytes of contiguous memory starting with the memory location where the `int x` was stored, but interpretted as `char`s. Something like this:

```
A☺ (4ö²☺pá \■▲áá ◄☺ ò☺H ☺   = ┗2 .    ■ á  =‖ ∞♣Ç     Ç  A ☺♀▄   ¶☺4|☺     ☻NÿDα
` ‖@Jî▲ [½2α  =αB
```

This is more likely to crash because it does 100 times as much writing to unallocated memory...

3. Compare and contrast the code below with that of problem # 2. Why is the cast needed here and why was it not needed in # 2?

```
#include <iostream>

int main() {
    int x = 97;
    void* y = &x;
    for(int i = 0; i < 100; ++i) {
        std::cout << *reinterpret_cast<char*>(y++);
    }
}
```

ANS: Here `y` is a pointer to `void` and so is not associated with any type. As such, we need to first reinterpret the type as a pointer to `char` and then dereference that. Again, if we're lucky, the program doesn't crash and we get something like this:

```
a   ♦   ↔ ( ( ( 4₧ßtö ( ²►@ ☺   p*5 á↑5     \ ( ┌î┌t
fpå■   ▲-ßtá§ßtá↑5     ◄(ßt☺    α²~ò↕@ ☺   H
```

4. Consider the `Matrix` class partially defined below.

```
class Matrix {
public:
    Matrix(int rows, int cols);
    Matrix(const Matrix& rhs);
    Matrix& operator=(const Matrix& rhs);
    double& operator()(int i, int j);
    const double operator()(int i, int j) const;
    void print() const;
private:
    int mRows;
    int mCols;
    int mSize;
    double* mData;
};
Matrix::Matrix(int rows, int cols)
: mRows(rows), mCols(cols), mSize(rows * cols), mData(new double[mSize]) { }
Matrix::Matrix(const Matrix& rhs)
: mRows(rhs.mRows), mCols(rhs.mCols), mSize(rhs.mSize)
{   mData = new double[mSize];
    std::memcpy(mData,rhs.mData,rhs.mSize * sizeof(double));
}
Matrix& Matrix::operator=(const Matrix& rhs) {
    if(&rhs == this) {
        return *this;
    }
    if(mSize == rhs.mSize)
        std::memcpy(mData,rhs.mData,mSize * sizeof(double));
    else {
        delete[] mData;
        mData = new double[rhs.mSize]();
        std::memcpy(mData,rhs.mData,mSize * sizeof(double));
    }
    mRows = rhs.mRows;
    mCols = rhs.mCols;
    mSize = rhs.mSize;
    return *this;
}
```

(a) On which line is the prototype for the copy constructor declared?
ANS: Line 4: `Matrix(const Matrix& rhs);`

(b) As it is, the constructor allocates memory for a Matrix, but does not initialize it. Where would you add code to intialize all matrix elements to 0 and what code would you add?
ANS: The constructor would be a natural place to fill in the zeros, like so:

```
Matrix::Matrix(int rows, int cols)
: mRows(rows),
  mCols(cols),
  mSize(rows * cols),
  mData(new double[mSize]()) {}
```

Note that this is exactly equivalent to lines 15 and 16, but with an added pair of empty parentheses. A little known feature of C++!

(c) How much of which part of memory would be allocated by the declaration `Matrix M(3,4)`?
ANS 8 bytes per double * 3 * 4 doubles = 96 bytes.

(d) A prototype for the overloaded parentheses operator is declared on line 6. Write a definition for this function which returns the element in the `i`th row and `j`th column of the matrix.
ANS:

```
double& Matrix::operator()(int i, int j)  {
    return mData[i * mCols + j];
}
```

(e) Write a definition for the class' `print()` method that uses the parentheses operator to access the element in row `i` and column `j`. Note that the "`const`" modifier of the method promises not to change any of the class' data members. Thus you will need a second definition for the overloaded parentheses operator whose return type is `const double` and which also promises not to modify any of the class' data members. ANS: Something like this:

```
const double Matrix::operator()(int i, int j) const {
    return mData[i * mCols + j];
}

void Matrix::print() const {
    for(int i = 0; i < mRows; ++i) {
        for(int j = 0; j < mCols; ++j)
            std::cout << std::setw(3) << operator()(i,j) << " ";
        std::cout << std::endl;
    }
}
```

(f) Matrix addition is defined by component-wise addition; that is, the sum of two matrices `A+B` is accomplished by adding `A(i,j)+B(i,j)`. Write a method for the class that overloads the addition operator to perform matrix addition.
ANS: Here's one way:

```
 Matrix& operator+(const Matrix& rhs) {
        if(rhs.mRows!=mRows || rhs.mCols!=mCols) {
            Matrix s(1,1);
            return s;
        }
        Matrix sum(mRows, mCols);
        for(int i = 0; i < mSize; ++i)
            sum.mData[i]=mData[i]+rhs.mData[i];
        return sum;
    }
```

(g) Modify the class to a template class for matrices of type `T`.
ANS: For this you need to add to the beginning of the class defintion code declaring it to be a `template` class like so:

```
template<typename T>
class Matrix{
```

Then wherever you see "`int`" as the default type of the matrix elements, change that to a "`T`":

```
template<typename T>
class Matrix{
public:
    Matrix(int rows, int cols);
    Matrix(const Matrix& rhs); // copy ctor.
    Matrix& operator=(const Matrix& rhs); // assign. ctor.
    T& operator()(int i, int j);
    const T operator()(int i, int j) const;
    Matrix<T>& operator+(const Matrix& rhs) {
        if(rhs.mRows!=mRows || rhs.mCols!=mCols) {
```

```
            Matrix s(1,1);
            return s;
        }
        Matrix<T> sum(mRows, mCols);
        for(int i = 0; i < mSize; ++i)
            sum.mData[i]=mData[i]+rhs.mData[i];
        return sum;
    }
    void print() const;
private:
    int mRows;
    int mCols;
    int mSize;
    T* mData;
};
template <typename T>
Matrix<T>::Matrix(int rows, int cols)
: mRows(rows),
  mCols(cols),
  mSize(rows * cols),
  mData(new T[mSize]()) {
    //for(int i = 0; i < mSize; ++i)
        //for(int j = 0; j < mCols; ++j)
            //*mData = 0.;
}
template <typename T>
Matrix<T>::Matrix(const Matrix<T>& rhs)
: mRows(rhs.mRows),
  mCols(rhs.mCols),
  mSize(rhs.mSize)
{   mData = new T[mSize];
    std::memcpy(mData,rhs.mData,rhs.mSize * sizeof(T));
}
template <typename T>
Matrix<T>& Matrix<T>::operator=(const Matrix<T>& rhs) {
    if(&rhs == this) {
        return *this;
    }
    if(mSize == rhs.mSize)
        std::memcpy(mData,rhs.mData,mSize * sizeof(T));
    else {
        delete[] mData;
        mData = new T[rhs.mSize]();
        std::memcpy(mData,rhs.mData,mSize * sizeof(T));
    }
    mRows = rhs.mRows;
    mCols = rhs.mCols;
    mSize = rhs.mSize;
    return *this;
}
```

```
template <typename T>
T& Matrix<T>::operator()(int i, int j)  {
    return mData[i * mCols + j];
}
template <typename T>
const T Matrix<T>::operator()(int i, int j) const {
    return mData[i * mCols + j];
}
template <typename T>
void Matrix<T>::print() const {
    std::cout<<"\nprint " << mRows << " by " << mCols << std::endl;
    for(int i = 0; i < mRows; ++i) {
        for(int j = 0; j < mCols; ++j)
            std::cout << std::setw(3) << operator()(i,j) << " ";
        std::cout << std::endl;
    }
}
```

(h) Write a driver that tests all the features of the `Matrix` class.
ANS: Here's something that does a whole bunch of testing (maybe not everything):

```
int main() {
    Matrix<int> M(3,4); //test constructor
    Matrix<int> N(3,4);
    M.print();
    for(int i = 0; i < 3; ++i) // test overloaded parentheses
        for(int j = 0; j < 4; ++j) {
            M(i,j) = 3*i+j;
            N(i,j) = 5*i-2*j;
        }
    std::cout << "\nM = \n"; M.print(); // test print
    std::cout << "\nN = \n"; N.print(); // test overload+
    std::cout << "\nM+N = \n";
    Matrix<int> P(3,4);
    P=M+N;  P.print();

    Matrix<float> F(2,2); // test assignment operator
    std::cout << "\nF = \n"; F.print();
    //test template of float
    Matrix<float> Q(2,2);
    for(int i = 0; i < 2; ++i)
        for(int j = 0; j < 2; ++j) {
            Q(i,j)=1./(i+1) +i/(i+j+1.);
            F(i,j)=1./(i+2) +i/(i+j+3.);
        }
   std::cout << "\nQ = \n";
   Q.print();
   F = F+Q;
   std::cout << "\nF+Q = "; F.print();
}
```

5. Consider the following code:

```
#include <iostream>
void gutz(char* s) {
    int i = 0, j = 0;
    for (; *s; ++s, ++i) {
        if (   tolower(*s)=='a' || tolower(*s)=='e'
            || tolower(*s)=='i' || tolower(*s)=='o'
            || tolower(*s)=='u') {
            j = 0;
            while(*s) {
                *s = *(s+1);
                ++j; ++s;
            }
            s -= j+1;
            --i;
        }
    }
    s -= i;
}
void print_array(char* s) {
    // write code here
}
void test(std::string s) {
    gutz(&s[0]);
    print_array(&s[0]);
    std::cout << "\n";
}
int main() {
    std::string s;
    while (std::cin>>s && s!="quit")
        test(s);
}
```

(a) Give a detailed description of what `gutz()` does and how it works. What type of input does it take? How does it process that input?
ANS: The function `gutz()` takes a `char` pointer and then uses pointer arithmetic in a `for` loop to check if a character in the c-string is either an upper or lower case vowel (aeiou). If it is, it copies each subsequent character over the previous character (thereby deleting the vowel from the string, counting how many characters it encounters in this way before reaching the `NUL` (`'\0'`) terminating character. When it reaches the `NUL` character, it sets the character pointer back to where it left off, and decrements the `for` loop counter by one, since you want to look and see if the very next `char` in the c-string is a vowel too. When the `for` loop terminates, the string pointer is set back to the first `char` in the c-string. The over-all effect is to delete all vowels (excepting 'y') from the c-string.

(b) In `test()` the argument passed to `gutz()` is `&s[0]`. What is this? Could the argument just as well have been simply `s`?
ANS: This is the address of the first element in the `std::string s`. You will get an error like "`cannot convert 'std::string (aka std::basic_string<char>)' to 'char*' for argument '1' to 'void print_array(char*)'`. The problem may be that a `std::string` is not required to be stored in contiguous memory.

(c) Write code for the body of `print_array()` that uses pointer arithmetic to print out the c-string passed to it.
ANS: Here is the body:

```
for (; *s; ++s)
    std::cout << *s;
```

6. The following recursive function prints the binary representation of a nonnegative integer:

```
void printBinary(int n) {
    if(n>1) printBinary(n/2);
    std::cout << n%2;
}
```

   (a) What is the base case here?
   ANS: The base case is when `n` is either 0 or 1.

   (b) Generalize this function into a recursive function `changeBase()` that accepts two integer parameters 'n' and 'base' and returns 'n' relative to a 'base' between 2 and 10;
   ANS: Here's the generalized recursive function and a driver to test it:

```
#include <iostream>

void printBase(int n, int base ) {
    if(n>base-1) printBase(n/base, base);
    std::cout << n%base;
}

int main() {
    int base, n;
    std::cout << "\nEnter the base and the decimal number to convert to that base: ";
    while(std::cin >> n >> base) {
        std::cout << n << " base " << base << " = ";
        printBase(n,base);
        std::cout << std::endl;
        std::cout << "\nEnter the base and the decimal number to convert to that base: ";
    }
}
```

7. (Coding exercise) Use a laboratory computer to write code meeting the following specifications.
   Of interest to mathematicians are 3-tuples of positive integers, (a, b, c) such that $c^2 = a^2 + b^2$. In the starter code below, we define a `struct Trip` to hold integers a, b, c together with a comparison operator ¡ for sorting. The program you'll develop here will be based on the idea that for any two relatively prime integers, $n < m$, that differ by an odd number $a = m^2 - n^2, b = 2mn, c = m^2 + n^2$ form a relatively unique `Triple`.
   Given a maximum value for c (say, 1000) the program will write the `Triple`s to a text file, something like this:

```
a: 3 b: 4 c: 5
a: 5 b: 12 c: 13
a: 15 b: 8 c: 17
...
a: 129 b: 920 c: 929
```

```
#include <vector>    // std::vector
#include <algorithm> // std::sort
#include <iostream>  // std::ostream, std::cout, std::endl
#include <fstream>
// Structure to hold triples
struct Trip {
  int a, b, c;
```

```
  bool operator< (Trip& t) { return c < t.c; } // Comparison operator for std::sort()
};
// Function to print triples
std::ostream& operator << (std::ostream& os, Trip& t) {
  //code here to output text such as "a: 3  b: 4 c: 5" and return the right sort of
    thing.
}
// Helper function to check if two integers are relatively prime
bool isRelPrime(int m, int n) {
  //for all ints from 2 to n/2
    // if m and n can be divided evenly by the same number, they are not relatively
    prime
      return false;
  return true;
}
void createTriples(int max, std::vector<Trip>& vT) {
  // for m between 2 and max
    // for n between 2 and m (n is the smaller one)
       // if m and n are relatively prime and difference is odd, we have a Trip
          // use the given formulae to create a Trip
          // When C is larger than max, return

         // otherwise, push new Trip onto the vector of Trips
      }
    }
  }
}
int main() {
  // Make a vector container to hold all triples
  // Fill the container with all triples
  // Sort the triples with sort(v.begin(),v.end())
  // Print the triples
  // Create of output file stream for "Triples.txt"
  for (std::vector<Trip>::iterator it = Triples.begin(); it != Triples.end(); ++it)
    // write the Trip to the file
}
```

ANS: Here it is!

```
// Structure to hold triples
struct Trip {
  int a, b, c;
  bool operator< (Trip& t) { return c < t.c; } // Comparison operator for std::sort()
};

// Function to print triples
std::ostream& operator << (std::ostream& os, Trip& t) {
  return os << "a: " << t.a << "\tb: " << t.b << "\tc: " << t.c << std::endl;
}

// Helper function to check if two integers are relatively prime
bool isRelPrime(int m, int n) {
  for (int i = 2; i < n/2; ++i)
    if (m%i==0 && n%i==0) // if they can be divided evenly by the same number, they
    are not relatively prime
      return false;
  return true;
}

void createTriples(int max, std::vector<Trip>& vT) {
```

```
21    for (int m = 2; m <= max; ++m)    {
        for (int n = 1; n < m; ++n)  { // n cannot be bigger than m
23        if ( isRelPrime(m,n) && ((m-n)%2==1))  { // if m and n are relatively prime and
      diff is odd, we have a Trip
            Trip temp;
25          temp.a = m*m - n*n; // using Euclid's formula to calculate a,b,c
            temp.b = 2*m*n;
27          temp.c = m*m + n*n;

29          if (temp.c > max) return; // When C is larger than max, return

31          vT.push_back(temp);              // otherwise add the triple to our set.
            sort(vT.begin(),vT.end());
33        }
        }
35    }
}
37
int main() {
39    // Make a vector container to hold all triples
    std::vector<Trip> Triples;
41
    // Fill the container with all triples
43  createTriples(1000, Triples);

45  // Sort the triples
    std::sort(Triples.begin(), Triples.end());
47
    // Print the triples
49  std::ofstream of("Triples.txt");
    for (std::vector<Trip>::iterator it = Triples.begin(); it != Triples.end(); ++it)
51    of << *it;
}
```