CS 7B Final Exam Solutions - Fall 2019 - Name:

With exception to the first matching problem, write responses to following on separate paper.

1. Write the number of the definition on the right next to the term it defines.

(a) **Generic programming** 14

(b) **invariant** 16

(c) **container** 18

(d) **recursion** 6

(e) **move** 1

(f) **copy** 5

(g) **template** 12

(h) **overload** 3

(i) **pointer** 2

(j) **reference** 11

(k) **class** 4

(l) **type** 15

(m) **dereference** 13

(n) **free Store** 7

(o) **byte** 17

(p) **constructor** 8

(q) **scope** 9

(r) **this** 10

(1) an operation that transfers a value from one object to another, leaving behind a value representing "empty."

(2) (1) a value used to identify a typed object in memory; (2) a variable holding such a value.

(3) Define two functions or operators with the same name but different argument (operand) types.

(4) A user-defined type that may contain data members, function members, and member types.

(5) An operation that makes two objects have values that compare equal..

(6) A function calling itself, it is hoped with different arguments so that the recursion eventually ends with a call for which the function doesn't call itself.

(7) A pool of unallocated heap memory given to a program that is used by the program for dynamic allocation during the execution of program.

(8) An operation that initializes an object. Typically establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor).

(9) The region of program text (source code) in which a name can be referred to.

(10) A pointer to object for which a member function is being called.

(11) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.

(12) A mechanism that allows a programmer to use types as parameters for a class or a function. The compiler then generates a specific class or function when we later provide specific types as arguments.

(13) An operator that will get the contents of a memory location.

(14) Writing code that works with a variety of types presented as arguments, as long as those argument types meet specific syntactic and semantic requirements..

(15) Something that defines a set of possible values and a set of operations for an object.

(16) Something that must be always true at a given point (or points) of a program; typically used to describe the state (set of values) of an object or the state of a loop before entry into the repeated statement.

(17) The basic unit of addressing in most computers.

(18) An object that holds elements (other objects).

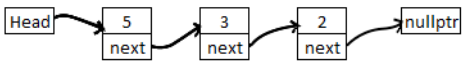
2. Like references to simple data types, we can have references to pointers. Consider the following complete program that employs this idea:

```
1  #include <iostream>
2
3  struct Node {
4      int data;
5      Node* next;
6  };
7  void push(Node*& alias, int x) { //alias references a pointer, another name for it
8      Node* node = new Node;
9      node->data = x;
10     node->next = alias;
11     alias = node;
12 }
13 void printList(struct Node* node) {
14   while (node != NULL) {
15     std::cout << node->data << " ";
16     node = node->next;
17   }
18 }
19 int main()  {
20   /* Start with the empty list */
21   Node* head = nullptr;
22   push(head, 2);
23   push(head, 3);
24   push(head, 5);
25   printList(head);
26 }
```

(a) What would the output of `std::cout head->data;` be if it were inserted at line 22?
ANS: This would crash the program since it would be dereferencing the `nullptr`, which results in a NullPointerException.

(b) Give a detailed description of what happens on line 8.
ANS: Memory is allocated on the heap for one `Node` object and the address of this node object is assigned to named `node`, which is, of course, a pointer to a `Node`.

(c) Draw a diagram of the list after line 24 is executed.
ANS:



(d) What is the output of the program?
ANS: 5 3 2

(e) Write a definition for the function void `add(Node*& alias, int x);` that will append a `Node` with `data = x` at the end of the list. **ANS:**

```
1  void add(Node*& alias, int x) {
2    Node* search = alias; //don't want to change head
3    while (search->next) search = search->next;//find the end of the list
4    Node* node = new Node;
5    node->data = x; //create new node and give it data
6    search->next = node; //append
7    node->next = nullptr;
8  }
```

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

```
1  class Matrix {
2  public:
3      Matrix(int rows, int cols);
4      Matrix(const Matrix& rhs);
5      Matrix& operator=(const Matrix& rhs);
6      double& operator()(int i, int j);
7      const double operator()(int i, int j) const;
8      void print() const;
9  private:
10     int mRows;
11     int mCols;
12     int mSize;
13     double* mData;
14 };
15 Matrix::Matrix(int rows, int cols)
16 : mRows(rows), mCols(cols), mSize(rows * cols), mData(new double[mSize]) { }
17 Matrix::Matrix(const Matrix& rhs)
18 : mRows(rhs.mRows), mCols(rhs.mCols), mSize(rhs.mSize)  {
19     mData = new double[mSize];
20     std::memcpy(mData,rhs.mData,rhs.mSize * sizeof(double));
21 }
22 Matrix& Matrix::operator=(const Matrix& rhs) {
23     if(&rhs == this) {
24         return *this;
25     }
26     if(mSize == rhs.mSize)
27         std::memcpy(mData,rhs.mData,mSize * sizeof(double));
28     else {
29         delete[] mData;
30         mData = new double[rhs.mSize]();
31         std::memcpy(mData,rhs.mData,mSize * sizeof(double));
32     }
33     mRows = rhs.mRows;
34     mCols = rhs.mCols;
35     mSize = rhs.mSize;
36     return *this;
37 }
```

(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();
}
```

4. 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;
```

5. 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: ";
    }
}
```