Transfer Bridge 2019 Preliminary Examination

<table>
<thead>
<tr>
<th>Section</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>30</td>
<td>25</td>
<td>100</td>
</tr>
<tr>
<td>Actual Points</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Assume the following variable declarations:

   int foo = 0;
   int *ptr = &foo;

Which of the following statements will change the value of foo to 1?
(1) ptr++;  
(2) foo++;  
(3) (*foo)++;  
(4) (*ptr)++;

(A) 1 and 2 only  
(B) 1 and 4 only  
(C) 2 and 4 only  
(D) 3 and 4 only

2. The hypotenuse function is correct, but the main function has a problem. Explain why it may not work, and show a way to fix it. Your fix must be to the main function only; you must not change the hypotenuse function in any way.

```cpp
#include <iostream>
#include <cmath>
using namespace std;

void hypotenuse(double leg1, double leg2, double* resultPtr) {
    *resultPtr = sqrt(leg1*leg1 + leg2*leg2);
}

int main() {
    double* p;
    hypotenuse(1.5, 2.0, p);
    cout << "The hypotenuse is " << *p << endl;
}
```

#include <iostream>
#include <cmath>
using namespace std;

void hypotenuse(double leg1, double leg2, double* resultPtr) {  
    *resultPtr = sqrt(leg1*leg1 + leg2*leg2);
}

int main() {  
    double* p;  
    hypotenuse(1.5, 2.0, p);  
    cout << "The hypotenuse is " << *p << endl;
}

```
3. For each of the following parts, write a single C++ statement that performs the indicated task. For each part, assume that all previous statements have been executed (e.g., when doing part c, assume the statements you wrote for parts a through d have been executed). For each part, do not use any variable names or string literals not mentioned in that part (e.g., if the part doesn’t mention fp or “tuna”, do not use fp or “tuna” in your answer).

a. Declare a pointer variable named fp that can point to a variable of type string.

b. Declare fish to be a 5-element array of strings.

c. Make the fp variable point to the last element of fish.

d. Make the string pointed to by fp equal to “yellowtail”, using the * operator.

e. Without using the fp pointer, and without using square brackets, set the fourth element (i.e., the one at position 3) of the fish array to have the value “salmon”.

f. Move the fp pointer back by three strings.

g. Using square brackets, but without using the name fish, set the third element (i.e., the one at position 2) of the fish array to have the value “loach”.

h. Without using the * operator or the name fish, but using square brackets, set the string pointed to by fp to have the value “eel”.

i. Using the == operator in the initialization expression, declare a bool variable named d and initialize it with an expression that evaluates to true if fp points to the string at the start of the fish array and to false otherwise.

j. Using the * operator in the initialization expression, but no square brackets, declare a bool variable named b and initialize it to true if the string pointed to by fp is equal to the string immediately following the string pointed to by fp, and false otherwise.
4. Given two strings \texttt{str1} and \texttt{str2}, \texttt{str1} is the permutation of \texttt{str2} if all the characters in \texttt{str1} appear in \texttt{str2} but in different order. For example, “12345” is the permutation of “54132”. Assume that only ‘1’ – ‘9’ will appear in the string. A student coded the following program to solve this problem, but there is something wrong. Please find all the bugs in this program.

```cpp
#include <iostream>
#include <string>
using namespace std;

bool isPermutation(string str1, string str2) {
    if(str1.size() != str2.size())
        return false;
    int i, j, counts[10];
    for(i=0;i<10;i++)
        counts[i] = 0;
    for(i=0;i<str1.size();i++)
        counts[ str1[i] - 0 ] ++; // (1)
    for(i=0;i<str1.size();i++)
        counts[ str2[i] - 0 ] --; // (2)
    for(i=0;i<10;i++)
        if(counts[i] != 0) // (4)
            return true; // (5)
    return false; // (6)
}

int main() {
    cout << isPermutation("12345","54321") << endl;
    cout << isPermutation("12345","98765") << endl;
    return 0;
}
```

The bugs are in?
(A) 123456
(B) 12356
(C) 1256
(D) 234
(E) 2356
(F) 256
(G) 356
(H) 3
(I) 12
(J) 34
5. What is the output produced by this program?

```cpp
bool contains(string s, char c){
    for (int k =0; k != s.size(); k++){
        if (s[k] == c)
            return true;
    }
    return false;
}

int main(){
    if (contains("Computer Science 32", '3'))
        cout << "W";
    if (!contains("Conan", 'x'))
        cout << "o";
    if (contains("OBrien", 'n'))
        cout << "W";
    if (contains("Start your CS 32 projects early!", ' '))
        cout << "!" << endl;
}
```

6. Suppose that a variable named \( b \) of type `bool` has already been given a value. What is the effect of the following statement?

\[
b = (b == false);
\]

a. It causes a compilation error.
b. It does something undefined during execution.
c. It causes \( b \) to have the value `false` regardless of \( b \)’s value before the statement was executed.
d. It always changes the value of \( b \).
e. It changes the value of \( b \) if and only if \( b \) had the value `true` just before the statement was executed.
7. You have been hired by ACME Transport, Inc., to write software to manage their fleet of Vehicles. The first piece of code you will write is a class that represents a gas tank. The **GasTank** class has the following interface:

1. A constructor that takes no parameters. When initialized, a gas tank has a capacity of 15 gallons and has zero gallons of gas in it.

2. An **addGas** member function that accepts a parameter of type double representing the number of gallons to attempt to add to the tank. The function adds gas to the tank and returns a double: the number of gallons actually added, which will be less than the requested amount if the tank becomes full. You may assume the parameter is guaranteed to be nonnegative.

3. A **useGas** member function that accepts a parameter of type double representing the number of gallons to attempt to remove from the tank (e.g., to send to the engine of a vehicle). The function removes gas from the tank and returns a double: the number of gallons actually removed, which will be less than the requested amount if the tank becomes empty. You may assume the parameter is guaranteed to be nonnegative.

   **a.** Write the declaration for the class **GasTank**, and implement the member functions. Using correct syntax, be sure to declare all function and data members, and use **public**, **private**, and **const** appropriately.
b. You will now write a class named `Vehicle` that represents a vehicle. Every Vehicle has a **GasTank**. All vehicles have a miles per gallon rating; for example, if a van has a 15.3 mpg rating, then it can travel 15.3 miles on each gallon of gas. The `Vehicle` class has the following interface:

1. A constructor that takes two parameters: a double representing the mpg rating of the vehicle, and a double specifying the number of gallons of gas that the vehicle starts with in its tank. You may assume the mpg is guaranteed to be positive and the initial number of gallons nonnegative.

2. An `addGas` member function that accepts a parameter of type double representing the number of gallons to attempt to add to the vehicle's gas tank. The function adds gas to the tank and returns a double: the number of gallons actually added, which will be less than the requested amount if the tank becomes full. You may assume the parameter is guaranteed to be nonnegative.

3. A `travel` member function that accepts a parameter of type double representing the number of miles to attempt to drive. The function removes the appropriate amount of gas from the vehicle's gas tank and returns a double: the number of miles actually traveled, which will be less than the requested number if the vehicle has too little gas in the tank. You may assume the parameter is guaranteed to be nonnegative.

4. A `milesTraveled` member function that takes no parameters and returns a double: the total number of miles the vehicle has traveled since it was created.

Write the declaration for the class `Vehicle`, and implement the member functions. Using correct syntax, be sure to declare all function and data members, and use `public`, `private`, and `const` appropriately.

On this page, write just the `Vehicle` class declaration. (You'll write the implementations of the member functions on the next page.)
Write the implementations of the `Vehicle` class's member functions on this page.
c. Write a function named **createCar** that takes no parameters. It dynamically allocates a new **Vehicle** that gets 20 mpg and starts off with 10 gallons of gas in it, then drives that vehicle 30 miles, and finally returns a pointer to that **Vehicle** to the function's caller.
8. Implement the class `Hotel`, as declared below, which keeps track of the reservation status of each room in a hotel. Each room can be `RESERVED`, `OCCUPIED`, or `EMPTY`, and this information is stored in a 2-dimensional array, where each row represents a floor, and each column represents a room. Each room is represented by an integer (e.g., 425). For example, the status of room 425 is stored in `m_rooms[4][25].

```cpp
const char RESERVED = 'R';
const char OCCUPIED = 'O';
const char EMPTY = 'E';

const int FLOORS = 20;
const int ROOMSPERFLOOR = 50;

class Hotel{
public:
    Hotel();
    bool reserve(int roomNum);
    bool cancel(int roomNum);
    bool checkIn(int roomNum);
    bool checkOut(int roomNum);
    int numEmpty(int floor) const;
private:
    char m_rooms[FLOORS][ROOMSPERFLOOR];
    // More private members here, if necessary.
}

Hotel::Hotel(){
    // EMPTY the rooms.
    for (int i = 0; i < FLOORS; i++)
        for (int j = 0; j < ROOMSPERFLOOR; j++)
            m_rooms[i][j] = EMPTY;
}

Implement other functions below.

bool Hotel::reserve(int roomNum){
    // TODO: If the room is EMPTY, set it to RESERVED, and return true.
    // In all other cases, do not change anything and return false.
}
bool Hotel::cancel(int roomNum){
    // TODO: If the room is RESERVED, set it to EMPTY, and return true.
    // In all other cases, do not change anything and return false.
}

bool Hotel::checkIn(int roomNum){
    // TODO: If the room is RESERVED, set it to OCCUPIED, and return true.
    // In all other cases, do not change anything and return false.
}

bool Hotel::checkOut(int roomNum){
    // TODO: If the room is OCCUPIED, set it to EMPTY, and return true.
    // In all other cases, do not change anything and return false.
}

int Hotel::numEmpty(int floor){
    // TODO: Return the number of empty rooms on the floor.
    // Return -1 if floor is invalid.
}

// Write helper functions down here if necessary.