CS 7B - Spring 2019 - Final Exam

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) **copy**____
   (b) **overload**_____
   (c) **container**_____
   (d) **pointer**_____
   (e) **reference**_____
   (f) **class**_____
   (g) **invariant**_____
   (h) **type**_____
   (i) **byte**_____
   (j) **constructor**_____
   (k) **scope**_____

   (1) (1) a value used to identify a typed object in memory; (2) a variable holding such a value.
   (2) an operation that transfers a value from one object to another, leaving behind a value representing "empty."
   (3) An operation that makes two objects have values that compare equal..
   (4) Define two functions or operators with the same name but different argument (operand) types.
   (5) A user-defined type that may contain data members, function members, and member types.
   (6) 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).
   (7) The region of program text (source code) in which a name can be referred to.
   (8) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.
   (9) Something that defines a set of possible values and a set of operations for an object.
   (10) 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.
   (11) The basic unit of addressing in most computers.
   (12) An object that holds elements (other objects).

2. Write the output of the following C++ code, which uses several kinds of C++ parameter passing:

```
int mystery(int a, int* b, int& c) {
    a++;
    (*b)++;
    c++;
    return a;
}

int main() {
    int a{0}, b{0}, c{0}, d{0};
    mystery(a, &b, c);
    cout << a << " " << b << " " << c << " " << d << endl;
    mystery(c, &d, a);
    cout << a << " " << b << " " << c << " " << d << endl;
    c = mystery(b, &a, d);
    cout << a << " " << b << " " << c << " " << d << endl;
    a = mystery(a, &a, a);
    cout << a << " " << b << " " << c << " " << d << endl;
}
```

3. Consider the following complete program

```
#include <iostream>
#include <vector>
#include <chrono>
using namespace std;

int f(int);
int f(int num, vector<int>& memoizeF);
int64_t f(int64_t num, int64_t *memoizeF);

int main() {
    int num;
    int64_t num64{ 0 };
    vector<int> memoizeV{ 0,1 };
    // -1 indicates that the value is not within the array
    while (cin >> num) {
        int64_t* memoizeP = new int64_t[num + 1]{ 0, 1 };
        for (int i = 2; i <= num; i++) {
            memoizeV.push_back(-1); // -1 means not done
            memoizeP[i] = -1;
        }
        num64 = num; //promotion
        auto t1 = std::chrono::system_clock::now();
        cout << "The time to compute fibonacci(" << num << ") = "
            << f(num) << " with dumb recursion: ";
        auto t2 = std::chrono::system_clock::now();
        cout << (t2 - t1).count() << endl;
        t1 = std::chrono::system_clock::now();
        cout << "The time to compute fibonacci(" << num << ") = "
            << f(num, memoizeV) << " with memoization vector: ";
        t2 = std::chrono::system_clock::now();
        cout << (t2 - t1).count() << endl;
        t1 = std::chrono::system_clock::now();
        cout << "The time to compute fibonacci(" << num << ") = "
             << f(num64, memoizeP) << " with memoization pointer: ";
        t2 = std::chrono::system_clock::now();
        cout << (t2 - t1).count() << endl;
        delete[] memoizeP;
    }
}

int64_t f(int64_t num, int64_t* memoizeP) {
    if (memoizeP[num] == -1) {
        memoizeP[num] = f(num-1, memoizeP) + f(num-2, memoizeP);
    }
    return memoizeP[num];
}

int f(int num, vector<int>& memoizeV) {
    if (memoizeV[num] == -1)    {
        memoizeV[num] = f(num-1, memoizeV) + f(num-2, memoizeV);
    }
    return memoizeV[num];
}

int f(int n) {
    if (n == 0 || n == 1) return n;
    else return f(n - 1) + f(n - 2);
}
```

(a) Describe the parameters that distinguish the three overloaded versions of `fibonacci()`.

(b) Describe what happens on line 13.

(c) Describe, in detail, the effect of the command on line 16.

(d) Why does line 16 need to be inside the `while`-loop but line 13 does not?

(e) Describe what the containers `memoizeF` and `memoize64` do that makes the computation of `f()` more efficient.

(f) A sampling of output for this program as the user enters 20,30,40 and 50 is shown below. Note the use of the `chrono` library to calculate the times of executions (measured in ticks).

```
20
The time to compute fibonacci(20) = 6765 with dumb recursion: 69501
The time to compute fibonacci(20) = 6765 with memoization vector: 46258
The time to compute fibonacci(20) = 6765 with memoization pointer: 40211

30
The time to compute fibonacci(30) = 832040 with dumb recursion: 974679
The time to compute fibonacci(30) = 832040 with memoization vector: 60426
The time to compute fibonacci(30) = 832040 with memoization pointer: 61168

40
The time to compute fibonacci(40) = 102334155 with dumb recursion: 121545885
The time to compute fibonacci(40) = 102334155 with memoization vector: 64019
The time to compute fibonacci(40) = 102334155 with memoization pointer: 68222

50
The time to compute fibonacci(50) = -298632863 with dumb recursion: 15006249361
The time to compute fibonacci(50) = -298632863 with memoization vector: 50684
The time to compute fibonacci(50) = 12586269025 with memoization pointer: 47437
```

Compare the growth in the time of computation for the three overloaded versions of `f()`.

(g) Why do the "dumb recursion" and "memoization vector" versions give an incorrect value for `f(50)` but the "memoization point" version gives the correct value?

4. Suppose the following C++ classes have been declared:

```
1  class C1 {
2  public:
3      void m1() {
4          cout << "C1 m1   ";
5      }
6      void m2() {
7          cout << "C1 m2   ";
8          m3();
9      }
10     void m3() {
11       cout << "C1 m3   ";
12       m1();
13     }
14 };
```

```
1  class C2 : public C1 {
2  public:
3      void m1() {
4          cout << "C2 m1   ";
5      }
6      void m2() {
7          C1::m3();
8          cout << "C2 m2   ";
9      }
10     void m3() {
11         cout << "C2 m3   ";
12         m1();
13     }
14 };
```

and that a client program declares the following variables:

```
C1* var1 = new C2();
C2* var2 = new C2();
```

Write the output that would be produced by each of the following calls, as it would appear on the console.

`var1->m1();` _____

`var1->m2();` _____

`var1->m3();` _____

`var2->m1();` _____

`var2->m2();` _____

`var2->m3();` _____

5. Assume that an integer and a pointer each takes 4 bytes. Also, assume that there is no alignment in objects.

   Predict the output following program. Why?

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

class Test {
    int x;
    int *ptr;
    int y;
};

int main() {
    Test t;
    cout << sizeof(t) << " ";
    cout << sizeof(Test *);
}
```

6. Write a program to print the area and perimeter of a triangle having sides of 3, 4 and 5 units by
   (a) creating a class named '`Triangle`' with a function to print the area and perimeter.
   (b) creating a class named '`Triangle`' with the constructor having the three sides as its parameters.

7. Print the sum, difference and product of two complex numbers by creating a class named '`Complex`' with separate functions for each operation whose real and imaginary parts are entered by the user.