ÖZYEĞİN UNIVERSITY

Faculty of Engineering
Department of Computer Science

CS 409/509
Advanced C++ Programming
Midterm Exam

Spring, 2017-18

Duration: 120 minutes

Name: _____

Student ID: _____ **Graduate or Undergraduate Student:** _____

Signature: _____

General Instructions:

- **DO NOT** start until you are told to do so.
- Check the **available time**. STOP writing when time is over.
- This is a **closed-book** exam. You **MAY NOT** use the textbook or any other supplementary material during the exam.
- You **MAY NOT** use a cheat sheet.
- You **MAY NOT** have your cell phone with you. Make sure to put it in your bag.
- You **MAY NOT** share anything (notes, pencils, erasers, etc.) with other people.
- You **MAY NOT** leave the exam room in the first **60** minutes; you **MAY NOT** enter the exam room after the **60**th minute.
- You **MAY NOT** enter the room if you leave (e.g., to take a restroom break).
- Read the questions **carefully**; make sure that you understand the problems well.
- Write legibly.

Grading:

1a (15)	**1b** (15)	**2** (10)	**3** (10)	**4** (10)	**5** (10)	**6** (10)	**7** (20)	**TOTAL** (100)

Q1) (30 pts)

```
namespace Q1
{
    template<typename T>
    struct Pair
    {
        T first, second;

        friend void print(const Pair<T>& pair)
        {
            std::cout << "[" << pair.first <<
                ", " << pair.second << "]" << std::endl;
        }

        // you will add your code here

    }; // end of Pair struct

} // end of Q1 namespace

int main()
{
    using namespace Q1;
    auto pair_int = Pair<int>(4, 5);
    print(pair_int);

    auto pair_float = Pair<float>(1.1f, 2.2f);
    print(pair_float);

    Pair<std::string> pair_str_1 = pair_int;
    print(pair_str_1);

    Pair<std::string> pair_str_2 = pair_float;
    print(pair_str_2);

    return 0;
}
```

CONSOLE OUTPUT

```
[4, 5]
[1.1, 2.2]
['4', '5']
['1.100000', '2.200000']
```

a) (15 pts) Make "`auto pair_int = Pair<int>(4, 5); print(pair_int);`" and "`auto pair_float = Pair<float>(1.1f, 2.2f); print(pair_float);`" lines work as shown in the console output by adding **only one function** to the Pair<T> class.

b) (15) Make "`Pair<std::string> pair_str_1 = pair_int; print(pair_str_1);`" and "`Pair<std::string> pair_str_2 = pair_float; print(pair_str_2);`" lines work as shown in the console output by adding **only one function** to the Pair<T> class.
Hint: for converting `thing` to std::string you can use *std::to_string(`thing`)* function. For instance, std::to_string(1.1f) returns "1.100000" string.

Q2) (10 pts) Write equivalent functor class named "Functor" for the following lambda function. It is instantiated as "Functor func(u, n);" as shown.

```
int u{5};
double n{0.1};
// auto func = [&u, n](const float f) { return f*u*n; }
Functor func(u, n);
return func(10); // returns 10.0f*5*0.1
```

Q3) (10 pts) Ignoring the now obsolete usage of new and delete operators what is the major problem with the following code. Give details. (Code compiles fine)

```
class A
{
public:
    A() { }
    ~A() { }
};

class B : public A
{
    int value = 0;
public:
    B() : A() { }
    ~B() { }
};

int main(int, char**)
{
    A* a = new B();
    delete a;
}
```

Q4) a) (5 pts) Can a member function be both *static* and *const* at the same time? Why, why not? Explain your answer in detail.

Q4) b) (5 pts) You can define a member function both *inline* and *virtual* at the same time, and it compiles. **But** can the compiler "inline" a virtual function? Why, why not?
Hint: "Inlining" means that the function call is skipped and fused with the caller code at compile-time.

Q5) (10 pts) What will be the output of the following program? Explain in detail.
Hint: it depends on the architecture! Give your answers both for 32bit and 64bit addressable space architectures. Assume that **int** is always 4 bytes. But a pointer's size must always depend on the architecture's addressable memory space.

```
#include <iostream>
struct A
{
    int data[2];

    A(int x, int y) : data{x, y} {}

    virtual void f() {}
};

int main(int, char**)
{
    A a(44, 55);
    auto* arr = (int*)&a;
    std::cout << arr[2] << std::endl; // prints the 3rd element pointed by the arr pointer

    return 0;
}
```

Q6) (10 pts)

a) (5 pts) Define a pointer to integer type where pointed integer cannot be changed at run-time.

b) (5 pts) Define a pointer to integer type where both the pointer itself and the pointed integer cannot be changed.

Q7) (20 pts) (FOR CS409 ***UNDERGRADUATE*** STUDENTS ONLY)

```
struct Foo
{
    int n;
    std::string s;
    float f;
};
```

Consider the above struct. Write a **compile-time** function named **get** that returns a reference to the n^{th} attribute of a Foo instance at **compile-time**.

This function doesn't need to be a generic function for all different types of structs. It can be customized for only Foo struct above.

Sample usage:

```
auto foo = Foo{10, "hello", 3.14};
int& N = get<0>(foo);            // N is a reference to the "n" attribute of foo variable
std::string& S = get<1>(foo);    // S is a reference to the "s" attribute of foo variable
float& F = get<2>(foo);          // F is a reference to the "f" attribute of foo variable
```

Q7) (20 pts) (FOR CS509 ***GRADUATE*** STUDENTS ONLY)

Write a function **FUNC** which takes a reference to any functor instance (call it **IN**) with a single int parameter as input, and returns a new functor instance (call it **OUT**) as output. OUT functor, when called with an int argument, should return a unique_ptr<std::string> that contains the string version of the returned value (enclosed by [and]) by the IN functor.

You **are not allowed** to use std::function!

Sample usage:

```
auto new_functor = FUNC([](int a) { return a+1; }); // define a functor instance (which takes an int) and pass it to FUNC as parameter

auto u_ptr = new_functor(5); // calls newly returned functor with input 5, and store whatever it returns in u_ptr

// remember that new_functor is supposed to return std::unique_ptr<std::string>
// therefore, u_ptr is a unique_ptr that owns a std::string instance. std::string instance is supposedly contains "[6]" as a string.

std::cout << *u_ptr << std::endl; // prints [6] to the console
```

Now, write a function called FUNC that suits to the above code. You **are not allowed** to use std::function!