Programming Languages **Week 10 Exercise Solution** ECE326, Fall 2020

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

1. It is not possible to cause infinite recursion when working with variadic template. T [✓ F]
2. C/C++ compiler cannot type check variable arguments in a variadic function. [✓ T] F
3. C++ template is an example of functional programming. [✓ T] F
4. Variadic template parameter pack only works for types and not constants. T [✓ F]
5. A template parameter pack can only appear at the end of template parameters. T [✓ F]

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following statements are true about C++ templates?

[✓ i.] You can have member function templates inside a class template.

ii. Member function templates can be specialized.

iii. Function templates can be partially specialized.

iv. When specializing a function template, you can change the number of parameters.

[✓ v.] When specializing a class template, you can completely redefine the entire class.

Question 3. Short Question

Given the following template definition:

```
template<typename T, int S>
T average(const T v[]) {
    T total = v[0];
    for (int i = 1; i < S; i++) {
        total += v[i];
    }
    return total/(T)S;
}
```

Your lab partner tries to use this template in the main function, but none of them work correctly. For each template usage, first say whether it is a compile-time or run-time error, then, explain the problem.

```
int main(int argc, const char * argv[]) {
    int len = ::atoi(argv[argc-1]);

    float fv[] = { 1.2, 2.2, 3.3 };                 // part 1
    cout << average<float, 2.0>(fv) << endl;

    int iv[] = { 2, 1, 7, 5 };                      // part 2
    cout << average<int, 9>(iv) << endl;

    std::string sv[] = { "hello", "world" };        // part 3
    cout << average<std::string, 2>(sv) << endl;

    long lv[] = { 7, 3, 9, 8, 2 };                  // part 4
    cout << average<long, len>(lv) << endl;

    return 0;
}
```

Part 1: compile-time. Template parameter S expects an int, not float.

Part 2: run-time. S is 9 but the array only has 4 elements, which will cause index out of bound error.

Part 3: compile-time. std:string does not support the add assign operator or the division operator.

Part 4: compile-time. Template argument only accepts compile-time constant, but len is a runtime variable.

Question 4. Programming Questions

1. In assignment 1, one of the tasks involved was building the client to send request packets to the server. Recall that the format of Value looks like this in a packet:

`type`	`size`	`buf`
4 bytes	4 bytes	*size* bytes

Integer Example:

`1`	`8`	`42`
4 bytes	4 bytes	8 bytes

Where type is one of:

```
enum ValueType { INTEGER = 1, FLOAT = 2, STRING = 3, };
```

And that a Row contains an array of values like this:

`count`	`value[0]`	`value[1]`	`...`	`value[count-1]`
4 bytes				

```
struct Request {
    Packet packet;
    void putvalue(long val);
    void putvalue(double val);
    void putvalue(const char *);

    template<typename... Args>
    void putrow(Args ... args);
} req;
```

```
class Packet {
    /* internal buffer */
public:
    //  use this function to
    //  add data to the end
    //  of the buffer
    void pack(const char * bytes,
              unsigned size);
};
```

a) Complete the implementation for serializing a 64-bit integer into a packet. You do not have to perform endianness conversion. The `type` field needs to be `INTEGER`, and the `size` field needs to `sizeof(long)`. Hint: use `packet.pack` to serialize data into the buffer.

```
void Request::putvalue(long val) {


     int temp = INTEGER;
     packet.pack((const char *)&temp, sizeof(int));
     temp = sizeof(long);
     packet.pack((const char *)&temp, sizeof(int));
     packet.pack((const char *)&val, temp);



}
```

b) Given the following sample usage:

```
req.putrow(123, 32.5, "hello world");  // packs 3 values, count = 3
long a = 234; const char * msg = "good luck";
req.putrow(a, msg);                    // packs 2 values, count = 2
```

Implement `putrow` for Request so that it is possible to pack any combination of integers, floats, and c-strings. Assume `putvalue` for int, double and c-string have also been implemented.

```
struct Request {
    /*
     * other declarations. Add your own helper methods if needed
     */

    template<typename ... Args>
    void putrow(Args ... args) {

        int temp = (int)sizeof...(Args);
        packet.pack((const char *)&temp, sizeof(int));
        putvalues(args ...);
    }

    // base case for recursion
    void putvalues() const {}

    template<class T, typename... Args>
    void putvalues(T val, Args ... args) {
        putvalue(val);
        putvalues(args ...);

    }
}
```

2. Write a variadic template function that calculates the population variance of a set of values. The return type of the template should always be double, but the template arguments should accept any numeric type that can be casted to a double.

```
See variance.cpp
```