TERM	COURSE NAME	COURSE CODE	VERSION
Fall-2019-Quiz-Lab-3	Object-Oriented Software Development using C++	OOP345	**A**

Code 1.0

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

template <typename T>
void fun(const T&x)
{
    static int count = 0;
    cout << "x = " << x << " count = " << count << endl;
    ++count;
    return;
}

int main()
{
   1. fun<int> (1);
   2. cout << endl;
   3. fun<int>(1);
   4. cout << endl;
   5. fun<double>(1.1);
   6. cout << endl;
   7. return 0;
}
```

1. Code 1.0, line 1 will print:

    a. x = 1 count = 0
    b. x = 1 count = 1
    c. x = 1.1 count = 0
    d. x = 1.1 count = 1
    e. x = 1.1 count = 2

2. Code 1.0, line 3 will print:

    a. x = 1 count = 0
    b. x = 1 count = 1
    c. x = 1.1 count = 0
    d. x = 1.1 count = 1
    e. x = 1.1 count = 2

3. Code 1.0, line 5 will print:

    a. x = 1 count = 0
    b. x = 1 count = 1
    c. x = 1.1 count = 0
    d. x = 1.1 count = 1
    e. x = 1.1 count = 2

Code 2.0

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

template<int n> struct funStruct
{
    static const int val = 2*funStruct<n-1>::val;
};

template<> struct funStruct<0>
{
    static const int val = 1 ;
};

int main()
{
        1.  cout << funStruct<3>::val << endl;
         return 0;
}
```

4. Code 2.0, line 1 will print:

    a. 8
    b. 4
    c. 2
    d. 1

Code 3.0

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

template <class T>
T max (T &a, T &b)
{
    return (a > b)? a : b;
}

template <>
int max <int> (int &a, int &b)
{
    cout << "Called ";
    return (a > b)? a : b;
}

int main ()
{
    int a = 10, b = 20;
    1.  cout << max <int> (a, b);
}
```

5. Code 3.0, line 1 will print:

    a. 20
    b. Called 20
    c. None of the above
    d. All of the above

Code 4.0

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

template <int i>
void fun()
{
   i = 20;
   cout << i;
}

int main()
{
   fun<10>();
   return 0;
}
```

6. Code 4.0, will result in:

   a. Printing 20
   b. Printing 10
   c. Compilation error
   d. None of the above

Code 5.0

```
template < class T >
class Base
{
    public:
        void set( const T& val) { data = val;}
        T    get(){ return data; };
    private:
        T data;
};
template < class T >
class Derived : public Base <T>
{
    public:
        void set( const T& val);
};
template< class T>
void Derived<T>::set( const T& v)
{
    Base<T>::set(v);
}
int main(){
    1. Derived<int> d;
    2. d.set(10);
    3. std::cout << d.get() << std::endl;
}
```

7. Code 5.0, will result in compilation error

   a. NO
   b. YES

8. Assuming Code 5.0 will compile, line 3 will print

   a. 10
   b. 20
   c. All of the above
   d. None of the above

Code 6.0

```
#include<iostream>
using namespace std;
int main()
{
        int x = 5;
        auto check = []() -> bool
        {
                if(x == 0)
                        return false;
                else
                        return true;
        };
        Cout <<. check() << endl;
        return 0;
}
```

9. Code 6.0, will result in compilation error

   a. YES
   b. NO

Code 7.0

```
#include<iostream>
using namespace std;
int main()
{
        int a = 5;
        auto check = [](int x)
        {
                if(x == 0)
                        return false;
                else
                        return true;
        };
        1. cout<<check(a)<<endl;
        return 0;
}
```

10. Code 7.0, line 1 print

    a. 0
    b. 1
    c. All of the above
    d. None of the above

Code 8.0

```
#include<iostream>
using namespace std;
int main()
{
        int a = 5;
        auto check = [=]()
        {
                a = 10;
        };
        check();
        cout<<"Value of a: "<<a<<endl;
        return 0;
}
```

11. Code 8.0, will result in compilation error

    a. YES
    b. NO

12. `[=](...)` captures:
    a. all non-local variables by value
    b. captures all non-local variables by reference
    c. All of the above
    d. None of the above
13. `[&](...)` captures:
    a. all non-local variables by value
    b. all non-local variables by reference
    c. All of the above
    d. None of the above
14. `[=,x,y](...)` captures:
    a. captures `x` and `y` by reference, all else by value
    b. captures `x` and `y` by value, all else by reference
    c. captures `x` by value and `y` by reference
15. `[&,x,y](...)` captures:
    a. captures `x` and `y` by reference, all else by value
    b. captures `x` and `y` by value, all else by reference
    c. captures `x` by value and `y` by reference
16. `[x,&y](...)` captures:
    a. captures `x` and `y` by reference, all else by value
    b. captures `x` and `y` by value, all else by reference
    c. captures `x` by value and `y` by reference
17. What does the function objects implement?
    a. Operator ()
    b. Operand <>
    c. None of the above
18. What is the advantage of function objects than the function call?
    a. It contains a state
    b. It is a type
    c. It contains a state & it is a type
    d. None of the above