Programming in Modern C++: Assignment Week 11

Total Marks : 25

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, Kharagpur – 721302
partha.p.das@gmail.com

Question 1

Consider the code segment (in C++11) given below. *[MSQ, Marks 2]*

```
int&& n1 = 10;        //LINE-1
auto&& n2 = n1;       //LINE-2

template<class T>
void fun1(T&& n);     //LINE-3

template<class T>
void fun2(std::vector<T>&& n);    //LINE-4
```

Identify the line/s where `&&` indicates a universal reference.

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: b), c)
Explanation:
Note that `&&` usually indicates rvalue reference. `&&` indicates a universal reference only where type deduction takes place.
At `LINE-1`, no type deduction takes place, therefore `&&` at `LINE-1` is just a rvalue reference, not a universal reference.
At `LINE-2`, like template `auto` also requires type deduction. Therefore `&&` at `LINE-2` indicates a universal reference.
At `LINE-3`, the template type parameter `T` requires type deduction. Thus, `&&` at `LINE-3` indicates a universal reference.
At `LINE-4`, the template type parameter `T` requires type deduction. However, since the form of function parameter is not `T&&` (it in form `std::vector<T>&&`), it indicates only rvalue reference.

Question 2

Consider the code segment (in C++11) given below. *[MCQ, Marks 2]*

```
#include<iostream>

void fun(const int& i){ std::cout << "lvalue " << i << std::endl; }
void fun(int&& i){ std::cout << "rvalue " << i << std::endl; }

template<typename T, typename U>
void wrapper(T&& n1, U&& n2){
    fun(n1);
    fun(n2);
    fun(std::move(n1));
    fun(std::move(n2));
    fun(std::forward<T>(n1));
    fun(std::forward<U>(n2));
}


int main(){
    int i = 10;
    wrapper(i, std::move(i));
    return 0;
}
```

What will be the output?

a)
```
lvalue 10
rvalue 10
lvalue 10
rvalue 10
lvalue 10
rvalue 10
```

b)
```
lvalue 10
lvalue 10
rvalue 10
rvalue 10
lvalue 10
rvalue 10
```

c)
```
lvalue 10
rvalue 10
rvalue 10
rvalue 10
lvalue 10
rvalue 10
```

d)
```
lvalue 10
lvalue 10
lvalue 10
rvalue 10
lvalue 10
rvalue 10
```

Answer: b)
Explanation:
Inside the `wrapper` function, the following calls:
`fun(n1);`
`fun(n2);`
both call the lvalue version of function `fun`. Thus the output will be:
`lvalue 10`
`lvalue 10`
Inside the `wrapper` function, the following calls:
`fun(std::move(n1));`
`fun(std::move(n2));`
both call the rvalue version of function `fun`. Thus the output will be:
`rvalue 10`
`rvalue 10`
Inside the `wrapper` function, the following calls:
`fun(std::forward<T>(n1));`
`fun(std::forward<T>(n2));`
call the lvalue version of function `fun` for `n1` and rvalue version for `n2`. Thus the output will be:
`lvalue 10`
`rvalue 10`

Question 3

Consider the following class (int C++11). *[MSQ, Marks 2]*

```
#include <iostream>

enum class Color {RED, GREEN, YELLOW} color;
enum class Signal {RED, GREEN, YELLOW} signal;
enum Tint {RED, GREEN, YELLOW} tint;

bool testRed(Color col){
    if(col == Color::RED)       //LINE-1
        return true;
    return false;
}

bool testGreen(Color col){
    if(col == Signal::GREEN)    //LINE-2
        return true;
    return false;
}

bool testYellow(Color col){
    if(col == YELLOW)           //LINE-3
        return true;
    return false;
}

int main() {
    if(testRed(Color::RED))
        if(testGreen(Color::GREEN))
            if(testYellow(Color::YELLOW))
                std::cout << "success";

    return 0;
}
```

Identify the statement/s which are true for the given program.

a) It generates compiler error at `LINE-1`

b) It generates compiler error at `LINE-2`

c) It generates compiler error at `LINE-3`

d) It generates the output `success`

Answer: b), c)
Explanation:
The statement `if(col == Color::RED)` compares between two `Color` type, which always compiles successfully.
The statement `if(col == Signal::GREEN)` compares between `Color` type with `Signal` type, which are not type castable. Thus it generates error.
The statement `if(col == YELLOW)` compares between `Color` type with `int`, which are not

type castable. Thus it generates error.
Since the code generates compiler error, it will not produce any compiler error.

Question 4

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>

template<typename T>
T findMax(T t){
    return t;
}

template<typename T, typename... Args>
T findMax(T t, Args... args){
    return ____________________________;    //LINE-1
}
int main() {

    std::cout << findMax(20, 30, 19, 60, 50);
    return 0;
}
```

Identify the appropriate return statement at `LINE-1` such that the function `findMax` finds out the maximum element from the given argument list (in this case the output is `60`)?

a) `t > findMax((args)...)`

b) `t < findMax((args)...)`

c) `t <= findMax((args)...)  ?  t :  findMax((args)...)`

d) `t > findMax((args)...)  ?  t :  findMax((args)...)`

Answer: d)
Explanation:
It recursively compares the first element of the parameter list `t` with the rest of the elements in `(args)...`. If `t` is greater than the maximum of `(args)...`, then `t` is the maximum element; else the maximum of `(args)...` (found recursively) is the maximum of the parameter list.

Question 5

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <algorithm>
#include <vector>

template<typename T>
void process(std::vector<T>& tVec){
    struct compare{
        bool operator()(T a, T b){ return a > b; }
    };
    sort(tVec.begin(), tVec.end(), compare());    //LINE-1
}

int main() {
    std::vector<int> iVec {20, 40, 60, 10, 50};
    process(iVec);
    for(int i : iVec)
        std::cout << i << " ";
    return 0;
}
```

What will be the output/error?

a) `10 20 40 50 60`

b) `60 50 40 20 10`

c) `20 40 60 10 50`

d) `compiler error at LINE-1:  compare is local`

Answer: b)
Explanation:
C++11 allows local declaration of functor within function scope, so it is not a compiler error. Furthermore, since the vector is passes to the as pass-by-reference, the effect of the sort would be reflected on the vector in `main` function. As per the logic implemented in functor `compare`, `sort` would sort the vector in descending order.

Question 6

Consider the following code segment (in C++11). *[MSQ, Marks 2]*

```
#include <iostream>
#include <vector>
#include <functional>

int main(){
    int sum = 0, multi = 0;
    ___________________________________ {    //LINE-1
        multi = x * y;
        return x + y;
    };
    auto result = process(10, 20);
    std::cout << "sum = " << result << ", multiplication = " << multi;
    return 0;
}
```

Identify the appropriate option(s) to fill in the blanks at `LINE-1` such that the output becomes `sum = 30, multiplication = 200`.

a) `auto process = [&multi](int x, int y)`

b) `auto process = [sum, multi](int x, int y)`

c) `auto process = [&](int x, int y)`

d) `auto process = [=](int x, int y)`

Answer: a), c)
Explanation:
Since the variable `multi` is modified within the lambda function it needs to be captured by reference. However, variable `sum` is not required to be captured as it is returned as result of the function.

Question 7

Consider the lambda function (in C++11) below. *[MCQ, Marks 2]*

```
auto process = [interval, &result](int val){
    result = val + interval;
    std::cout << "(" << val << ", " << interval << ") = " << result;
};
```

Identify the correct option that define the equivalent Closure object for the above lambda function.

a)
```
struct process_s {
    int& interval;
    int result;
    process_s(int& r) : result(r) { }
    void operator()(int val) const {
        result = val + interval;
        std::cout << "(" << val << ", " << interval << ") = " << result;
    };
};
auto process = process_s(result);
```

b)
```
struct process_s {
    int& interval;
    int result;
    process_s(int& i, int r) : interval(i), result(r) { }
    void operator()(int val) const {
        result = val + interval;
        std::cout << "(" << val << ", " << interval << ") = " << result;
    };
};
auto process = process_s(interval, result);
```

c)
```
struct process_s {
    int interval;
    int& result;
    process_s(int i, int& r) : interval(i), result(r) { }
    void operator()(int val) const {
        result = val + interval;
        std::cout << "(" << val << ", " << interval << ") = " << result;
    };
};
auto process = process_s(interval, result);
```

d)
```
struct process_s {
    int interval;
    process_s(int i) : interval(i) { }
    void operator()(int val, int& result) const {
        result = val + interval;
        std::cout << "(" << val << ", " << interval << ") = " << result;
    };
};
auto process = process_s(interval);
```

Answer: b)
Explanation:
For a λ-expression, the compiler creates a functor class with:

- data members:
  - a value member each for each value capture (`interval`)
  - a reference member each for each reference capture (`result`)
- a constructor with the captured variables as parameters
  - a value parameter each for each value capture
  - a reference parameter each for each reference capture
- a public inline const function call `operator()` with the parameters of the lambda as parameters, generated from the body of the lambda
- copy constructor, copy assignment operator, and destructor

Question 8

Consider the following code segment (in C++11). *[MCQ, Marks 2]*

```
#include <iostream>
#include <string>

class Data{
    public:
        explicit Data(int _d1) : d1(_d1){ std::cout << "ctor-1" << " "; } //ctor-1
    private:
        int d1 { 10 };
};

class DataPair : public Data{
    public:
        DataPair() = default;
        explicit DataPair(int _d1) : DataPair(_d1, 0.0f) { //ctor-2
            std::cout << "ctor-2" << " ";
        }
        explicit DataPair(int _d1, double _d2)
                : DataPair(_d1, _d2, "unknown") { //ctor-3
            std::cout << "ctor-3" << " ";
        }
        DataPair(int _d1, float _d2)
                : DataPair(_d1, _d2, "unknown"){ //ctor-4
            std::cout << "ctor-4" << " ";
        }

    private:
        DataPair(int _d1, float _d2, std::string _d3)
                : Data(_d1), d2(_d2), d3(_d3){ //ctor-5
            std::cout << "ctor-5" << " ";
        }
        double d2 { 0.0 };
        std::string d3 { "empty" } ;
};

int main(){
    DataPair data(100);
    return 0;
}
```

What will be the output?

a) `ctor-1 ctor-5 ctor-3 ctor-2`

b) `ctor-1 ctor-5 ctor-4 ctor-2`

c) `ctor-2 ctor-4 ctor-5 ctor-1`

d) `ctor-5 ctor-1 ctor-3 ctor-2`

Answer: b)
Explanation:

The statement `DataPair data(100);` invokes constructor `ctor-2`. The delegations from one constructor to another would be as follows: `ctor-2 -> ctor-4` (since the second argument is `float` type), `ctor-4 -> ctor-5 -> ctor-1`. Thus, the output will be in reverse order that is `ctor-1 ctor-5 ctor-4 ctor-2`.

Question 9

Consider the following code segment (in C++). *[MSQ, Marks 2]*

```
#include <iostream>
#include <string>

struct StringData{
    explicit StringData(const std::string& _d) : d(_d) {}
    std::string d;
};

struct IntData{
    explicit IntData(const int& _d) : d(_d) {}
    operator int() { return d; }
    explicit operator int*() const { return nullptr; }
    explicit operator std::string() const { return "test"; }
    explicit operator StringData() const { return StringData("test"); }
    int d;
};

int main(){
    IntData data(100);
    int i1 = data;                                      //LINE-1
    int i2 = static_cast<int>(data);                    //LINE-2
    int *i3 = data;                                     //LINE-3
    int *i4 = static_cast<int*>(data);                  //LINE-4
    std::string s1 = static_cast<std::string>(data); //LINE-5
    StringData s2 = static_cast<std::string>(data);  //LINE-6
    StringData s3 = static_cast<StringData>(data);   //LINE-7
    return 0;
}
```

Identify the line(s) that will generate compiler error.

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

e) `LINE-5`

f) `LINE-6`

g) `LINE-7`

Answer: c), f)
Explanation:
Since casting to `int*` operator is explicit `LINE-3` would generate error. At `LINE-6`, `static_cast` would convert the `data` to `string` type, but the LHS of the expression is of `StringData` type. Therefore, it is wrong.

Programming Questions

Question 1

Consider the program below (in C++11).

- Fill in the blank at `LINE-1` with appropriate template declaration.
- Fill in the blank at `LINE-2` with an appropriate parameter for function `createPerson`.
- Fill in the blank at `LINE-3` to complete the `return` statement.

The program must satisfy the given test cases. *Marks: 3*

```
#include <iostream>
class person{
    public:
        virtual void show() = 0;
};

class Employee : public person {
    private:
        double salary;
    public:
        Employee(const double& _salary) : salary(_salary){
            std::cout << "lvalue" << " ";
        }
        Employee(double&& _salary) : salary(_salary){
            std::cout << "rvalue" << " ";
        }
        void show(){ std::cout << salary << " "; }
};

class Player : public person {
    private:
        int rank;
    public:
        Player(const int& _rank) : rank(_rank){
            std::cout << "lvalue" << " ";
        }
        Player(int&& _rank) : rank(_rank){
            std::cout << "rvalue" << " ";
        }
        void show(){ std::cout << rank << " "; }
};

____________________________        //LINE-1
T createPerson(__________){         //LINE-2
    return ____________________;    //LINE-3
}
int main() {
    double s;
    int r;
    std::cin >> s >> r;
```

```
    auto p1 = createPerson<Employee>(s);
    auto p2 = createPerson<Employee>(std::move(s));
    auto p3 = createPerson<Player>(r);
    auto p4 = createPerson<Player>(std::move(r));
    std::cout << std::endl;
    p1.show();
    p2.show();
    p3.show();
    p4.show();
    return 0;
}
```

Public 1

```
Input:
10000.5 10
Output:
lvalue rvalue lvalue rvalue
10000.5 10000.5 10 10
```

Public 2

```
Input:
50000 3
Output:
lvalue rvalue lvalue rvalue
50000 50000 3 3
```

Private

```
Input:
66000.5 2
Output:
lvalue rvalue lvalue rvalue
66000.5 66000.5 2 2
```

Answer:

`LINE-1:` `template<typename T, typename U>`

or

`LINE-1:` `template<class T, class U>`

`LINE-2:` `U&& param`

`LINE-3:` `T(std::forward<U>(param))`

Explanation:

This code is a typical example of factory method. Since the function return type and the parameters both are template parameter type, at `LINE-1` the template must be declared as:

`template<typename T, typename U>`

or

`template<class T, class U>`

The parameter for `createPerson` must be universal reference type which can be declared as:

`LINE-2:` `U&& param`

Depending on the instantiation of parameter type `T` the function `createPerson` return an object of `Employee` or `Player` type, which can be done as:

`return T(std::forward<U>(param))`

Question 2

Consider the following program (in C++14).

- Fill in the blank at `LINE-1` with an appropriate template declaration for the function `apply`.
- Fill in the blank at `LINE-2` with an appropriate header for function `apply`.
- Fill in the blank at `LINE-3` with an appropriate return statement for function `apply`.

The program must satisfy the sample input and output. *Marks: 3*

```
#include <iostream>
template<typename T>
class Data{
    private:
        T _d;
    public:
        Data(T data) : _d(data){}
        T getData() const{
            return _d;
        }
};

template<typename T, typename U>
struct Adder{
    U operator()(std::ostream& os, Data<T>&& d1, Data<U>&& d2){
        os << "rvalue version: ";
        return (d1.getData() + d2.getData());
    }
    U operator()(std::ostream& os, const Data<T>& d1, const Data<U>& d2){
        os << "lvalue version: ";
        return (d1.getData() + d2.getData());
    }
};

_____________________________    //LINE-1
_____________________________________________ {    //LINE-2
    return _______________________________;    //LINE-3
}

int main() {
    int i;
    double d;
    std::cin >> i >> d;
    Data<int> d1(i);
    Data<double> d2(d);

    std::cout << apply(std::cout, Adder<int, double>(), d1, d2);
    std::cout << std::endl;
    std::cout << apply(std::cout, Adder<int, double>(), std::move(d1), std::move(d2));
    return 0;
}
```

Public 1

```
Input:
10 15.5
Output:
lvalue version: 25.5
rvalue version: 25.5
```

Public 2

```
Input:
-1 5.3
Output:
lvalue version: 4.3
rvalue version: 4.3
```

Private

```
Input:
-10 3.14
Output:
lvalue version: -6.86
rvalue version: -6.86
```

Answer:
In C++11

```
LINE-1: template<typename F, typename... T>
```

or

```
LINE-1: template<class F, class... T>
LINE-2: auto apply(std::ostream& os, F&& func, T&&... args) ->
                            decltype(func(os, args...))
```

or in C++14

```
LINE-2: decltype(auto) apply(std::ostream& os, F&& func, T&&... args)
LINE-3: func(os, std::forward<T>(args)...)
```

Explanation:
At `LINE-1` the template for function `apply` must be declare as:
`template<typename F, typename... T>`
or
`template<class F, class... T>`
At `LINE-2` header for function `apply` must be:
`auto apply(std::ostream& os, F&& func, T&&... args) -> decltype(func(os, args...))`
in C++11,
or
`decltype(auto) apply(std::ostream& os, F&& func, T&&... args)` in C++14.
At `LINE-3` the return statement should be:
`return func(os, std::forward<T>(args)...);`

Question 3

Consider the following program that implements a recursive lambda function to find out minimum element from a given vector.

- Fill in the blank at `LINE-1` to declare the signature of `RecMin` as `std::function`.
- Fill the blank at `LINE-2` to complete the definition of lambda function `RecMin`.

The program must satisfy the sample input and output. *Marks: 3*

```
#include <iostream>
#include <vector>
#include <functional>

int main(){
    _________________________________;    //LINE-1
    _________________________________ {   //LINE-2
        return n == 1 ? tVec[0] : tVec[n - 1] < RecMin(tVec, n - 1) ?
            tVec[n - 1] : RecMin(tVec, n - 1);
    };
    auto Print = [&RecMin](std::vector<int> tVec) {
        std::cout << RecMin(tVec, tVec.size());
    };
    int n, m;
    std::vector<int> vec;
    std::cin >> n;
    for(int i = 0; i < n; i++){
        std::cin >> m;
        vec.push_back(m);
    }
    Print(vec);
    return 0;
}
```

Public 1

```
Input: 6
9 3 6 2 1 7
Output: 1
```

Public 2

```
Input: 5
10 -20 30 -40 50
Output: -40
```

Private

```
Input: 4
-1 1 0 -2
Output: -2
```

Answer:

```
LINE-1:  std::function<int(std::vector<int>, int)> RecMin
LINE-2:  RecMin = [&RecMin](std::vector<int> tVec, int n) -> int
```

Explanation:

At `LINE-1`, we can use `std::function` to declare the signature of `RecMin` as:
`std::function<int(std::vector<int>, int)> RecMin`
At `LINE-2` to complete the definition of lambda function `RecMin` is as follows:

```
RecMin = [&RecMin](std::vector<int> tVec, int n) -> int {
        return n == 1 ? tVec[0] : tVec[n - 1] < RecMin(tVec, n - 1) ?
            tVec[n - 1] : RecMin(tVec, n - 1);
    };
```