Programming in Modern C++: Assignment Week 11

Total Marks : 20

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

September 27, 2023

Question 1

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

```
template<typename T1>
class RefType{
    public:
        void f1(T1&& n){}               //LINE-1
        template<typename T2>
        void f2(T2&& n){}               //LINE-2
        template<typename T3>
        void f3(std::list<T3>&& n);     //LINE-3
    private:
        auto&& 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), d)
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 during function call (the type deduction takes place during class instantiation), therefore `&&` at `LINE-1` is just a rvalue reference, not a universal reference.
At `LINE-2`, the template type parameter `T2` requires type deduction. Thus, `&&` at `LINE-2` indicates a universal reference.
At `LINE-3`, the template type parameter `T3` requires type deduction. However, since the form of function parameter is not `T&&` (it in form `std::list<T>&&`), it indicates only rvalue reference. At `LINE-4`, like template `auto` also requires type deduction. Therefore `&&` at `LINE-2` indicates a universal reference.

Question 2

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

```
#include <iostream>

template<typename T>
class Data{
    public:
        Data() {};
        Data(T i) : i_(i){ }
        Data(const Data& ) = delete;
        Data& operator=(const Data& ) = default;
        Data(Data&& ) = default;
    private:
        T i_;
};
int main(){
    Data<int> d1;
    Data<int> d2(30);                //LINE-1
    Data<int> d3 = d2;               //LINE-2
    Data<int> d4 = std::move(d2);    //LINE-3
    d1 = std::move(d2);              //LINE-4
    return 0;
}
```

Which of the following line/s generate/s compiler error/s?

a) `LINE-1`

b) `LINE-2`

c) `LINE-3`

d) `LINE-4`

Answer: b)
Explanation:
Since the copy constructor of class `Data` are explicitly deleted, `LINE-2` generates compiler error. Please note that though the move assignment operator is not defined, the statement at `LINE-4` will not generate error because if move assignment operator is not present it calls copy assignment operator by default.

Question 3

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

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

void process(std::vector<std::string>& v){
    struct cmp{
        bool operator()(std::string x, std::string y){ return x.length() > y.length(); }
    };
    sort(v.begin(), v.end(), cmp());
}

int main() {
    std::vector<std::string> v {"orange", "banana", "apple", "kiwi"};
    process(v);
    for(auto it : v)
        std::cout << it << " ";
    return 0;
}
```

What will be the output/error?

a) `apple banana kiwi orange`

b) `kiwi apple orange banana`

c) `orange banana apple kiwi`

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

Answer: c)
Explanation:
C++11 allows local declaration of functor within function scope, so it is not a compiler error. Furthermore, since the vector is passed 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 `cmp`, `sort` would sort the vector in descending order of the length of the strings.

Question 4

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

```
#include <iostream>

class data {
    public:
        data(){}
        explicit data(int i) : i_(i) { }
    protected:
        int i_ { 0 };
};

class data_pair: public data {
    public:
        explicit data_pair(double j) : j_(j) { }
        ------------------------------    //LINE-1
        void show(){
            std::cout << "(" << i_ << ", " << j_ << ")" << " ";
        }
    protected:
        double j_ { 0.0 };
};

int main(){
    data_pair d1(10);
    data_pair d2(10.5);
    d1.show();
    d2.show();
    return 0;
}
```

Choose the appropriate option/options to fill in the blank at `LINE-1` such that output becomes `(10, 0) (0, 10.5)` .

a) `data_pair(double i) :  i_(i) { }`

b) `using data::data;`

c) `data_pair(int i) :  data(i) { }`

d) `data_pair(int i) :  i_(i) { }`

Answer: b), c)
Explanation:
The statement `data_pair d1(10);` requires to initialize `i_` which means it needs to call the constructor of `data` within `data_pair`. Therefore, at `LINE-1` we fetch the base class `data` constructor as `using data::data;`.
Alternatively, we can have another constructor in `data_pair` which forward the call to the base class.

Question 5

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

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

int main(){
    double sum = 0.0, avg = 0.0;
    ____________________________________ {    //LINE-1
        for(auto it : v)
            sum += it;
        return sum / v.size();
    };
    std::vector<double>vd {10.6, 20.4, 30.4, 40.2};
    avg = stat(vd);
    std::cout << "avg = " << avg << ", sum = " << sum;
    return 0;
}
```

Identify the appropriate option(s) to fill in the blanks at `LINE-1` such that the output becomes `avg = 25.4, sum = 101.6`.

a) `auto stat = [&sum](std::vector<double> v)`

b) `auto stat = [sum, avg](std::vector<double> v)`

c) `auto stat = [=](std::vector<double> v)`

d) `auto stat = [&](std::vector<double> v)`

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

Question 6

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

```
#include <iostream>

class employee{
    public:
        explicit employee() : employee(0) {}    //LINE-1
        explicit employee(const int emp_id) :
                employee(emp_id, defaultSalary) {} //LINE-2
        explicit employee(const double salary) : employee(0, salary) {}    //LINE-3
        explicit employee(int emp_id, double salary) : emp_id_{emp_id},
                            salary_{salary} {}  //LINE-4
        friend std::ostream& operator<<(std::ostream&, const employee&);
    private:
        int emp_id_ {-1};
        double salary_ {0.0};
        static constexpr double defaultSalary {20000.00};
};

std::ostream& operator<<(std::ostream& os, const employee& e){
    os << "[" << e.emp_id_ << " - " << e.salary_ << "]" << ", ";
    return os;
}

int main(){
    employee e1;
    employee e2(10);    //LINE-5
    employee e3(60000.0);  //LINE-6
    employee e4(20, 75000.5);
    std::cout << e1 << e2 << e3 << e4;
    return 0;
}
```

What will be the output/error?

a) `[0 - 20000], [10 - 20000], [-1 - 60000], [20 - 75000.5],`

b) `[0 - 20000], [10 - 20000], [0 - 60000], [20 - 75000.5],`

c) `compiler error at LINE-5:  ambiguous call s2(10)`

d) `compiler error at LINE-6:  ambiguous call s3(60.0)`

Answer: b)
Expanation:
The statement at `LINE-5` call the default constructor at `LINE-1`, which delegates the call to the parameterized constructor at `LINE-2`, which further delegates the call to the parameterized constructor at `LINE-4`.

Question 7

Consider the code segment (C++14) given below. *[MCQ, Marks 2]*

```
#include <iostream>

template<typename T> T pi = T(22L)/7;
int main(){
    pi<int> = 100;
    auto r1 = [](auto(deg)) { return pi<decltype(deg)> * deg / 180; }(360);
    auto r2 = [](auto(deg)) { return pi<double> * deg / 180; }(360);
    auto r3 = [](auto(deg)) { return pi<decltype(deg)> * deg / 180; }(360.0);
    auto r4 = [](auto(deg)) { return pi<int> * deg / 180; }(360.0);
    std::cout << r1 << ", " << r2 << ", " << r3 << ", " << r4;
    return 0;
}
```

What will be the output?

a) `6, 6.28571, 6.28571, 6`

b) `6, 6.28571, 6.28571, 200`

c) `200, 6.28571, 6.28571, 200`

d) `200, 6.28571, 6.28571, 6.28571`

Answer: c)
Explanation:
In the expression: `[](auto(deg)) return pi<decltype(deg)> * deg / 180; (360)` , the inferred type of `deg` is `int`. So, the result is `200`.
In the expression: `[](auto(deg)) return pi<double> * deg / 180; (360)` , the `pi<double>` is `3.1415926535897932385L`. So, the result is `6.28571`.
In the expression: `[](auto(deg)) return pi<decltype(deg)> * deg / 180; (360.0)` , the inferred type of `deg` is `double`. So, the result is `6.28571`.
In the expression: `[](auto(deg)) return pi<int> * deg / 180; (360.0)` , the `pi<int>` is `100`. So, the result is `200`.

Question 8

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

```
auto process = [&hv, fact](std::vector<int> v){
        for(auto it : v)
            hv.push_back(it * fact);
    };
std::vector<int> vc{10, 20, 30};
process(vc);
```

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

a)
```
struct process_s {
      std::vector<double> hv;
      double& fact;
      std::vector<int> v;
      process_s(std::vector<int> _v) : v(_v) { }
      void operator()(std::vector<double> _hv, double& _fact) const {
          for(auto it : v)
              hv.push_back(it * fact);
      }
  };
  std::vector<int> vc{10, 20, 30};
  auto process = process_s(vc);
```

b)
```
struct process_s {
      std::vector<double> hv;
      double& fact;
      process_s(std::vector<double> _hv, double& _fact) : hv(_hv), fact(_fact) { }
      void operator()(std::vector<int> v) const {
          for(auto it : v)
              hv.push_back(it * fact);
      }
  };
  auto process = process_s(hv, fact);
```

c)
```
struct process_s {
      std::vector<double>& hv;
      double fact;
      process_s(std::vector<double>& _hv, double _fact) : hv(_hv), fact(_fact) { }
      void operator()(std::vector<int> v) const {
          for(auto it : v)
              hv.push_back(it * fact);
      }
  };
  auto process = process_s(hv, fact);
```

d)
```
struct process_s {
      std::vector<double> hv;
      double fact;
      process_s(std::vector<double> _hv, double _fact) : hv(_hv), fact(_fact) { }
      void operator()(std::vector<int> v) const {
```

```
        for(auto it : v)
            hv.push_back(it * fact);
    }
};
auto process = process_s(hv, fact);
```

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

- data members:
  - a value member each for each value capture (`fact`)
  - a reference member each for each reference capture (`hv`)
- a constructor with the captured variables as parameters (`fact, hv`).
  - 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 9

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

```
enum class SIGNAL {RED, GREEN, YELLOW};
enum class COLOR {RED, GREEN, YELLOW, BLUE};
enum HOUSE {RED, GREEN, YELLOW, BLUE};

bool is_red(SIGNAL type){
    if(type == SIGNAL::RED)         //LINE-1
        return true;
    return false;
}

bool is_green(SIGNAL col){
    if(col == COLOR::GREEN)         //LINE-2
        return true;
    return false;
}

bool is_yellow(SIGNAL col){
    if(col == YELLOW)               //LINE-3
        return true;
    return false;
}
```

Identify the statement/s which are true for the above code segment.

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) There is no error in the given code segment

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

Programming Questions

Question 1

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

- Fill in the blanks at `LINE-1` and `LINE-3` with appropriate template definitions.
- Fill in the blanks at `LINE-2` and `LINE-4` to complete the return statements for `product` functions.

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

```
#include <iostream>

_____________________________                    //LINE-1
double product(T num){ ____________________ }    //LINE-2

_____________________________                    //LINE-3
double product(T num, Tail... nums){
    return num * _____________________;          //LINE-4
}

int main(){
    int a, b, c;
    double d, e, f;
    std::cin >> a >> b >> c;
    std::cin >> d >> e >> f;
    std::cout << product(a, b, c) << " ";
    std::cout << product(d, e, f) << " ";
    std::cout << product(a, b, c, d, e, f);
    return 0;
}
```

Public 1

```
Input:
2 3 4
2.3 3.4 4.6
Output:
24 35.972 863.328
```

Public 2

```
Input:
10 20 30
1.5 2.3 -4.5
Output:
6000 -15.525 -93150
```

Private

```
Input:
10 -11 12
2.3 2.5 2.7
```

```
Output:
-1320 15.525 -20493
```

Answer:

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

or

```
LINE-1:  template <class T>
LINE-2:  return num;
LINE-3:  template <typename T, typename...  Tail>
```

or

```
LINE-3:  template <class T, class...  Tail>
LINE-4:  product(nums...)
```

Explanation:

At `LINE-1`, the definition of the simple template is:

```
template <typename T>
```

or

```
template <class T>
```

, and at `LINE-3` the return statement the function `product` is:

```
return num;
```

At `LINE-3`, the definition of the veriadic template is:

```
template <typename T, typename...  Tail>
```

or

```
template <class T, class...  Tail>
```

, and at `LINE-4` the complete the return statment of function `product` as:

```
return num * product(nums...);
```

Question 2

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

- Fill in the blank at `LINE-1` with appropriate template declaration.
- Fill in the blanks at `LINE-2` with an appropriate universal reference type parameter for constructor of `class derived` and an the appropriate call forwarding to the base class constructor.

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

```
#include <iostream>

class base {
    public:
        base(const int& n) : n_(n * 10){ std::cout << "lvalue : " << n << ", "; }
        base(int&& n) : n_(n * 20) { std::cout << "rvalue : " << n << ", "; }
    protected:
        int n_;
};

class derived : public base {
    public:
        ____________________                                    //LINE-1
        ____________ : ______________________ { }               //LINE-2
        void show(){ std::cout << n_ << " "; }
};
int main(){
    int i;
    std::cin >> i;
    derived obj1(i);
    derived obj2(std::move(i));
    obj1.show();
    obj2.show();
    return 0;
}
```

Public 1

```
Input: 10
Output: lvalue : 10, rvalue : 10, 100 200
```

Public 2

```
Input: 50
Output: lvalue : 50, rvalue : 50, 500 1000
```

Private

```
Input: 5
Output: lvalue : 5, rvalue : 5, 50 100
```

Answer:

`LINE-1:  template<typename T>`

or

```
LINE-1:  template<class T>
LINE-2:  derived(T&& n) :  base(std::forward<T>(n))
```

Explanation:

At `LINE-1` the template must be declared as:

```
template<typename T>
```

or

```
template<class T>
```

At `LINE-2`, universal reference type parameter for constructor of `class derived` and the call forwarding to `base` class can be done as:

```
derived(T&& n) :  base(std::forward<T>(n))
```

Question 3

Consider the following program that implements a recursive lambda function to find the sum of the digits of an input integer.

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

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

```
#include<iostream>
#include<functional>
int main() {
    ____________________________________;    //LINE-1

    revPrint = _________________________ {     //LINE-2
        if (n == 0)
            return 0;

        return n % 10 + revPrint(n /= 10);
    };

    int a;
    std::cin >> a;
    std::cout << revPrint(a);
}
```

Public 1

```
Input: 12345
Output: 15
```

Public 2

```
Input: 4343
Output: 14
```

Private

```
Input: 1045
Output: 10
```

Answer:
`LINE-1:  std::function<int(int)> revPrint`
`LINE-2:  [&revPrint](int n) -> int`
Explanation:
At `LINE-1`, we can use `std::function` to declare the signature of `revPrint` as:
`std::function<int(int)> revPrint`
At `LINE-2` to complete the definition of lambda function `revPrint` is as follows:

```
revPrint = [&revPrint](int n) -> int {
        if (n == 0)
            return 0;
```

```
    return n % 10 + revPrint(n /= 10);
};
```