Programming in C++: Assignment Week 8

Total Marks : 25

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

March 17, 2019

Question 1

Consider the program below: *[MCQ, Marks 2]*

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

void f(int val) {
    cout << val << " ";
    if (val)
        throw val;
    else
        throw "value is zero";
}

int main() {
    try {
        // statement-1
    }
    catch (int i) {
        cout << i << endl;
    }
    catch (const char* s) {
        cout << s << endl;
    }

    return 0;
}
```

What will be the outputs in consecutive two runs if `statement-1` is replaced by `f(-1)` and `f(0)` respectively?

a) `-1 -1`
`value is zero value is zero`

b) `-1 value is zero`
`0 value is zero`

c) `-1 -1`
`0 0`

d) `-1 -1`
`0 value is zero`

Answer: d)
Explanation:
For the invocation `f(-1)`, `val` is set to `-1`. Hence, the condition `if (val)` is true and it throws `int` type exception. The exception will be caught by the catch block `catch (int i)`. So, it prints '`-1 -1`'.
Invocation of `f(0)` set `val` to 0. Hence, the condition `if (val)` is false and it throws the exception '`value as zero`'. The exception will be caught in the catch block `catch (const char* s)`. The catch block will print the value is `s` as '`value is zero`'. So, the output is '`0 value is zero`'.

Question 2

What will be the output of the following program? *[MCQ, Marks 2]*

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

class One {
public:
    One() { cout << "1" << " "; }
};

class Two : public One {
public:
    Two() { cout << "2" << " "; }
};

class Three : public Two {
public:
    Three() { cout << "3" << " "; }
};

int main() {
    try {
        throw Three();
    }
    catch (One&) {
        cout << "1" << endl;
    }
    catch (Two&) {
        cout << "2" << endl;
    }
    catch (Three&) {
        cout << "3" << endl;
    }

    return 0;
}
```

a) `1 2 3 1`

b) `3 2 1 1`

c) `1 2 3 3`

d) `3 2 1 3`

Answer: a)
Explanation:
The statement `throw Three();` creates an object of class `Three`. For creating object of class `Three`, the following class constructors are need to be invoked implicitly in the given order: class `One`, `Two` and `Three`. Hence, the output will be `1 2 3`. Next, the type of exception thrown is of type `Three`. But, in catch sequence first catch block if for `One&`, which is of parent class type. Hence, it will execute the first catch block and output is `1`.

Question 3

Consider the following program. *[MCQ, Marks 2]*

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

int main() {
    try{
        try {
            throw 3.14;
        }
        catch (int) {
            cout << "int" << endl;
        }
    }
    catch (char) {
        cout << "char" << endl;
    }
    catch (float) {
        cout << "float" << endl;
    }
    catch (...) {
        cout << "all" << endl;
    }

    return 0;
}
```

What will be the output / error?

a) `int all`

b) `float`

c) `all`

d) Exception: terminate called after throwing an instance of '`double`'

Answer: c)
Explanation:
The statement `throw 3.14;` throws an exception of type `double` (as `3.14` is a `double` literal). Nautrally, this will not be caught by `catch (int)` and will be thrown again.

This again will not be caught by the first `catch (char)` clause, or the next `catch (float)` clause. Hence, the last catch-all clause `catch (...)` will catch and the output is '`all`'.

Question 4

What will be the output of the following program? *[MCQ, Marks 2]*

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

class SpecificException : public exception {
public:
    const char * what() const throw () {
        return "it is a specific exception";
    }
};

class MoreSpecificException : public SpecificException {
public:
    const char * what() const throw () {
        return "it is a more specific exception";
    }
};

int main() {
    try {
        throw MoreSpecificException();
    }
    catch (SpecificException& e) { // Clause 1
        cout << "Clause 1: " << e.what() << endl;
    }
    catch (MoreSpecificException& e) { // Clause 2
        cout << "Clause 2: " << e.what() << endl;
    }
    catch (exception& e) { // Clause 3
        cout << "Clause 3: " << e.what() << endl;
    }

    return 0;
}
```

a) `Clause 1:  it is a specific exception`

b) `Clause 1:  it is a more specific exception`

c) `Clause 2:  it is a more specific exception`

d) `Clause 3:  Unknown exception`

Answer: b)
Explanation:
In the given code, we have two user-defined exceptions in a hierarchy rooted at `exception` – `MoreSpecificException` ISA `SpecificException` ISA `exception`. Hence, the thrown exception `MoreSpecificException` will be caught by the first clause as it is a specialization of class `SpecificException`. This rules out (c) and (d).

As the exception object is passed by reference in `catch (SpecificException& e)`, '`e`' is a `SpecificException` type reference to a `MoreSpecificException` object. Further, as we know, `exception::what()` is virtual:

```
class exception {
public:
    exception () noexcept;
    exception (const exception&) noexcept;
    exception& operator= (const exception&) noexcept;
    virtual ~exception();
    virtual const char* what() const noexcept;
}
```

Hence, `e.what()` will dynamically bind to the object type `MoreSpecificException:: what()` and the output will be as in (b).

Note that the output will be (a), if the exception object is passed by value in the first catch clause as `catch (SpecificException e)`. If passed by value the thrown object will be sliced and its base (`class SpecificException`) part will be copy constructed as '`e`. Hence, `e.what()` will bind to `SpecificException::what()`.

Question 5

Consider the following statements and identify the correct TRUE/FALSE pairs.

[MCQ Mark 2]

1. Generic catch handler must be placed after all the catch handlers
2. Functions called from within a try block may also throw an exception
3. A constructor should not throw any exception
4. In nested try blocks, there is no need to specify catch handler for inner try block. Outer catch handler is sufficient for the program

a) 1-TRUE, 2-TRUE, 3-TRUE, 4-TRUE

b) 1-TRUE, 2-TRUE, 3-TRUE, 4-FALSE

c) 1-TRUE, 2-TRUE, 3-FALSE, 4-TRUE

d) 1-FALSE, 2-FALSE, 3-TRUE, 4-TRUE

Answer: b)
Explanation:
Except 4^{th} statement all other statements are TRUE. So Option b) is the correct answer

Question 6

Fill in the blank space with appropriate option such that the `findMax()` function can find out the maximum value from the given two arrays. *[MCQ, Marks 2]*

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

____________________ // fill in the blank
T findMax(T arr[], int n) {
    T m = min;
    for (int i = 0; i < n; i++)
    if (arr[i] > m)
        m = arr[i];

    return m;
}

int main() {
    int arr1[] = { 18, 30, 35, 22 };
    int n1 = sizeof(arr1) / sizeof(arr1[0]);

    char arr2[] = { 's', 't', 'b', 'u', 'p' };
    int n2 = sizeof(arr2) / sizeof(arr2[0]);

    cout << findMax<int, -999>(arr1, n1) << endl;
    cout << findMax<char, 0>(arr2, n2);

    return 0;
}
```

a) `template <class T>, int min`

b) `template <class T, int min>`

c) `template <class T, int>`

d) `template <T, int min>`

Answer: b)
Explanation:
In `template <class T, int min>`, `T` is the generic type and `min` is a variable of type `int`. For the invocation '`findMax<int, -999>`', `min` will be assigned to `-999` and for invocation '`findMax<char, 0>`', `min` will be assign to `0`.

Question 7

Consider the following function definition. *[MCQ, Marks 2]*

```
template <typename T>
T Max(T x, T y) {
    return x > y ? x : y;
}
```

What will be the output / error associated with the following function calls?

i) `Max(10, 20);`

ii) `Max(10.9, 3.14);`

iii) `Max(10.8, 10);`

a) All calls are erroneous as type is not specified during invocation

b) i) 20 ii) 10.9 and iii) 10.8

c) i) 20 but ii) and iii) are error as by default only integer type can be passed in a template function

d) i) 20 ii) 10.9 but iii) is error as no matching for `Max(double, int)`

Answer: d)
Explanation:
For `Max(10, 20)`, the output is the maximum value (of `int` type) 20.
For `Max(10.9, 3.14)`, the output is the maximum value (of `double` type) 10.9.
For `Max(10.8, 10)`, it gives error: no matching function for call to '`Max(double, int)`' as it is ambiguous as to whether `T` is an `int` or a `double` type.

Question 8

What is the output of the following program? *[MCQ, Marks 2]*

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

template<class T = int, class U = int>
class Test {
    T x; U y;
public:
    Test(T t, U u) : x(t), y(u) { }
    void display() { cout << x << "," << y << endl; }
};

int main() {
    Test<char, int> t1('a', 10);
    t1.display();

    Test<> t2('a', 10);
    t2.display();

    return 0;
}
```

a) `a, 10`
   `a, 10`

b) `97, 10`
   `97, 10`

c) `a, 10`
   `97, 10`

d) Error: type is not specified

Answer: c)
Explanation:
In the statement `Test<char, int> t1('a', 10);`, the first argument is explicitly defined as `char` and second as `int`. Hence, the output is `a 10`.

In the statement `Test<> t2('a', 10);`, both the arguments are of `int` type. Hence, the argument '`a`' will be treated as `97` (the ASCII value of '`a`') and the output is `97 10`.

Question 9

Choose the right option/s to fill in the blanks (at `line-1`, `line-2`, and `line-3`) in program below such that the output will be: *[MSQ, Marks 2]*

```
x = 20, y = 10
x = b, y = a
```

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

_____________  // line-1: declare class-template
class swapper {
    T _a, _b;
public:
    swapper(T& a, T& b) :_a(a), _b(b) { }
    void swap() {
        T _t;
        _t = _a;
        _a = _b;
        _b = _t;
    }
    void display() { cout << "x = " << _a << ", y = " << _b << endl; }
};

int main() {
    int a = 10, b = 20;
    _____________ pair1(a, b); // line-2: declare swapper object
    pair1.swap();
    pair1.display();

    char c1 = 'a', c2 = 'b';
    _____________ pair2(c1, c2); // line-3: declare swapper object
    pair2.swap();
    pair2.display();

    return 0;
}
```

a) `line-1`: `template<class T>`, `line-2`: `swapper<int, int>`, `line-3`: `swapper<char, char>`

b) `line-1`: `template<class T>`, `line-2`: `swapper<int>`, `line-3`: `swapper<char>`

c) `line-1`: `template<typename T>`, `line-2`: `swapper<int, int>`, `line-3`: `swapper<char, char>`

d) `line-1`: `template<typename T>`, `line-2`: `swapper<int>`, `line-3`: `swapper<char>`

Answer: b), d)
Explanation:
As number of template arguments is one, so option a) and c) can be discarded (as number of template arguments are two).

The keyword `class` and `typename` can be used interchangeably in template. So, both the options b) and d) are correct.

Programming Questions

Question 1

Fill in the blanks with proper code to get the output as per the test cases. *Do not change any other part of the code.* *[Marks 2]*

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

class sample {
    int a1;
    int a2;
public:
    sample() { cin >> a1 >> a2; }

    // Complete the compute function header
    ____________________ { a1 = -a1; a2 = -a2; }

    // Complete the function header of display.
    // Do not disturb "cout"
    ____________________ { cout << a1 << " " << a2 << endl; }
};

class num :public sample {
    int n1;
    int n2;
public:
    num() { cin >> n1 >> n2; }
    void compute() { n1 = -n1; n2 = -n2; }
    void display() { cout << n1 << " " << n2; }
};

int main() {
    sample *ptr, ob1;
    num ob2;

    ptr = &ob1;
    ptr->compute();
    ptr->display();

    ptr = &ob2;
    ptr->compute();
    ptr->display();

    return 0;
}
```

Public Test Case 1

```
Input: 100 200 100 200 1000 2000
Output: -100 -200 -1000 -2000
```

Public Test Case 2

```
Input: 500 300 200 100 22 33
Output: -500 -300 -22 -33
```

Private Test Case

```
Input: 900 800 80 90 9000 8000
Output: -900 -800 -9000 -8000
```

Answer:
`virtual void compute()`
`virtual void display()`
Explanation:
The function names and return types are trivial – unless `void`, each function would have needed a `return` statement. Without the `virtual` the calls on `obj2` will not get dispatched to the member functions in `num` class.

Question 2

Fill in the blanks with proper code so that input and output of the test cases would satisfy. *Do not change any other part of the code.* *[Marks 3]*

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

class base {
protected:
    int b_;
public:
    base(int b): b_(b) { cout << b_ * b_ << " "; }
    virtual void display() { }
};

class derived : public base {
    int d_;
public:
    derived(int b, int d) : base(b), d_(d) { cout << d_ * d_ << " "; }
    void display();
};

class appObj {
    appObj(int x, int y) { cout << x + y << " " << endl; }

    ____________________;
};

void derived::display() {
    __________ obj1(b_, d_);
}

int main() {
    int m, n;
    cin >> m >> n;

    base *ptr = new derived(m, n);

    ptr->display();

    return 0;
}
```

Public Test Case 1

```
Input: 2 3
Output: 4 9 5
```

Public Test Case 2

```
Input: 3 7
Output: 9 49 10
```

Private Test Case

```
Input: 4 6
Output: 16 36 10
```

Answer:
`friend void derived::display()`
`appObj`
Explanation:
First note that from the test cases the output in terms of the inputs (`m` and `n`) is:

```
m*m n*n m+n
```

Now, consider the first test case. We have the following values from it.

```
m <- 2
n <- 3
base::b_ <- 2
derived::d_ <- 3
```

This will satisfy the first two output values (`4 9`) from the constructor of `base` followed by the constructor of `derived` arising from `new derived(m, n)` in function `main()`.

Next, `derived::display()` will be called from `ptr->display()` in `main()`. Hence, the last output can be printed only if the constructor of `appObj` is called. This is possible from the construction of `obj1(b_, d_)` which in this case is `obj1(2, 3)`. Hence, the second blank should be filled up by `appObj`.

Finally, we note that the constructor `appObj::appObj(int, int)` is private in class `appObj`. Hence, `derived::display()` will be allowed to access (call) it only if this is a `friend` function in `appObj`. Hence, the first blank should be filled as `friend void derived::display()`.

Similar analysis holds good for other test cases as well.

Question 3

Fill in the blanks in the program below to match the test cases. *Do not change any other part of the code.* *Marks: 3*

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

class A { protected: int ai;
public:
    A(int i) : ai(i) { }
    __________ void f() = 0;       // Fill the blank or Remove blank -- LINE 1
    __________ void g() {          // Fill the blank or Remove blank -- LINE 2
        ++ai;
    }
};

class B : public A { protected: int bi;
public:
    B(int i) : A(i), bi(i) { }
    void f() { cout << ai << bi; }

    void g() {
        ++ai;
        A::g();
    }
};

class C : public B { int ci;
public:
    C(int i) : B(i), ci(i) {}
    void f() { cout << ai << bi << ci; }

    void g() {
        ++ai; ++bi;
        B::g();
    }
};

int main() {
    int x = 3;
    int y;
    cin >> y;

    A *p[] = { new B(x), new C(y) };
    for (int i = 0; i < sizeof(p) / sizeof(A*); ++i) {
        p[i]->g();
        p[i]->f();
    }

    return 0;
}
```

Public Test Case 1

```
Input: 2
Output: 53532
```

Public Test Case 2

```
Input: 23
Output: 53262423
```

Private Test Case

```
Input: 4
Output: 53754
```

Answer:
`virtual`
`virtual`
Explanation:
In `LINE 1`, we clearly have the header for a pure virtual function. Hence, we need to fill up with `virtual`.

For `LINE 2`, let us assume that we remove the blank. The calls from within the `for` loop then are: `A::g() B::f() A::g() C::f()`. So for input `2`, the output will be `43` from `B::f()` and `322` from `C::f()`. This does not match the test case.

Next for `LINE 2`, let us assume that the fill up is `virual`. The calls from within the `for` loop then are: `B::g() A::g() B::f() C::g() B::g() A::g() C::f()`. So for input `2`, the output will be `53` from `B::f()` and `532` from `C::f()`. This matches the test case.

Finally, the same may be validated for the other test case too.

Question 4

Fill in the blanks in the program below to match the test cases. *Do not change any other part of the code.* *Marks: 2*

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

class Employee {
public:
    virtual void salary() { }
};

class Manager : public Employee {
    int sal;
public:
    Manager(int a) :sal(a) {}
    void salary() { cout << sal; }
};

class Programmer : public Employee {
    int sal, bon;
public:
    Programmer(int a, int b) : sal(a), bon(b) {}
    void salary() { cout << sal << ":"; }
    void bonus() { cout << bon << ":"; }
};

void paycheck(Employee *ep) {
    Programmer *pp = ____________________;
    if (pp)
        pp->bonus();
    else
        ep->salary();
}

int main() {
    int Psal, Pbon, Msal;
    cin >> Psal >> Pbon >> Msal;

    Employee *eptr = new Programmer(Psal, Pbon);
    paycheck(eptr);

    eptr = new Manager(Msal);
    paycheck(eptr);

    return 0;
}
```

Public Test Case 1

```
Input:
5000
300
10000
Output: 300:10000
```

Public Test Case 2

```
Input:
12000
550
25000
Output: 550:25000
```

Private Test Case

```
Input:
8000
450
15000
Output: 450:15000
```

Answer

```
dynamic_cast< Programmer* >(ep);
```

Question 5

Analyze the following program and fill in the blanks at `line-1` and `line-2` such that the given test cases would pass. *Marks: 2*

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

//declare a user-defined exception
class NegativeValException : public __________ { // line-1
public:
    virtual const char* what() const throw() {
        return "A Negative value";
    }
};

int main() {
    int i;
    cin >> i;

    try {
        if (i < 0)
            //throw the exception object
            throw _____________________; // line-2
        else
```

```
            cout << i << " is a +ve value";
    }
    catch (NegativeValException e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

Public 1

```
Input:  10
Output: 10 is a +ve value
```

Public 2

```
Input:  -20
Output: A Negative value
```

Private 1

```
Input:  30
Output: 30 is a +ve value
```

Private 2

```
Input:  -40
Output: A Negative value
```

Answer:
`line-1:  exception`
`line-2:  throw NegativeValException()`
Explanation:
An user-defined exception should be inherited from `exception` class. Hence, the `line-1` is:
`class NegativeValException :  public exception { }`

Now in `line-2`, if the value of `i` is negative, then we throw the exception as:
`throws NegativeValException()`.

Question 6

The following program adds integer and float values. Fill in the banks in (`Line-1 and Line-2`) such that the program would match the given sample input and output. *Marks: 3*

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

template<class T>
class Adder {
    T n1, n2;
public:
    Adder(T _n1, T _n2) :n1(_n1), n2(_n2) { }
    T Add();
};

____________________       // line-1: Declare the Template
T _________________::Add() { // line-2: Fill with the correct Template signature
    return n1 + n2;
}

int main() {
    int n1, n2;
    float f1, f2;
    cin >> n1 >> n2;
    cin >> f1 >> f2;

    Adder<int> obj1(n1, n2);
    Adder<float> obj2(f1, f2);

    cout << obj1.Add() << " " << obj2.Add() << endl;

    return 0;
}
```

Public 1

```
Input:  10 15
        2.14 4.56
Output: 25 6.7
```

Public 2

```
Input:  54 18
        10.99 18.12
Output: 72 29.11
```

Private

```
Input:  18 1
        7.18 9.19
Output: 19 16.37
```

Answer:
`line-1:` `template<class T>` or `typename T`
`line-2:` `Adder<T>`
Explanation:
The template function `Add()` outside the class must be defined with explicit template signature, that is `template<class T>` and the template parameter has to be defined as `T`, hence `T Adder<T>::Add()`.

Question 7

In the following program, fill in the banks (in `line-1` and `line-2`) as per the given instruction so that the test cases would pass. *Marks: 2*

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

template <class T>
class MyClass {
    T data1, data2;
public:
    MyClass(T _data1, T _data2) : data1(_data1), data2(_data2) {}
    T max() { return data1 > data2 ? data1 : data2; }
};


_____________   // line-1: Create explicit specialization for C-string
_____________ { // line-2: Write the required class header
    char *data1, *data2;
public:
    MyClass(char* _data1, char* _data2) :
        data1(strdup(_data1)), data2(strdup(_data2)) { }

    char* max() {
        if (strcmp(data1, data2) > 0)
            return data1;
        return data2;
    }
};

int main() {
    char str1[20];
    char str2[20];

    int n1;
    int n2;

    cin >> str1 >> str2;
    cin >> n1 >> n2;

    MyClass<char*> obj1(str1, str2);
    cout << obj1.max() << endl;

    MyClass<int> obj2(n1, n2);
    cout << obj2.max() << endl;

    return 0;
}
```

Public 1

```
Input:  test data
        82 46
Output: test
        82
```

Public 2

```
Input:  fun function
        34 56
Output: function
        56
```

Private

```
Input:  hello world
        100 200
Output: world
        200
```

Answer:

```
line-1:  template<>
line-2:  class MyClass<char*>
```

Explanation:
The first version of `MyClass` is a generic version and the second version is specialized version for C-string (`char*`). In `line-1`, we use `template<>` construct, to create explicit specialization class for `MyClass`. The type of data for which specialization is being created is placed inside the angle brackets following the class name.