Programming in C++: Assignment Week 8

Total Marks : 20

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

October 13, 2020

Question 1

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

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

void fun(int test) {
    try {
        test ? throw test : throw "zero ";
    }
    catch (int i) {                    //LINE-1
        cout << "Caught: " << i << " ";
    }
}
int main() {
    try{
        fun(1);
        fun(2);
        fun(0);
        fun(3);
    }
    catch (const char *str) {            //LINE-2
        cout << "CaughtString ";
    }
    return 0;
}
```

What will be the output?

a) `Caught:  1 Caught:  2 CaughtString Caught:  3`

b) `Caught:  1 Caught:  2 CaughtString`

c) `Caught:  1 Caught:  2 CaughtString zero Caught:  3`

d) `Caught:  1 Caught:  2`

Answer: b)
Explanation:
The first call `fun(1);`, results in `throw test;` where `test` = 1. It will be caught within the function `fun()`, and prints 1.
Then the call `fun(2);`, results in `throw test;` where `test` = 2. It will be caught within the function `fun()`, and prints 2.
Then the call `fun(0);`, results in `throw "zero "`. But, in `fun()` there is no `catch` block to catch `const char*` type. Hence, the exception will be forward to the `main()` function. In `main()`, `catch` block at `LINE-2` handle the exception, and prints `CaughtString` . Then `main()` returns and `fun(3)` is never called.
Hence the output is option b).

Question 2

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

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

namespace cust_error {
    class error { };
    class spec_error : public error { };
    class unknown_error : public error { };
    void f() { throw unknown_error(); }
};

int main() {
    try {
        cust_error::f();
    }
    catch (cust_error::spec_error&) {        // LINE-1
        cout << "specific error" << endl;
    }
    catch (cust_error::error&) {             // LINE-2
        cout << "error" << endl;
    }
    catch (cust_error::unknown_error&) {     // LINE-3
        cout << "unknown error" << endl;
    }
    catch (...) {                            // LINE-4
        cout << "default" << endl;
    }
    return 0;
}
```

What will be the output?

a) `specific error`

b) `error`

c) `unknown error`

d) `default`

Answer: b)
Explanation:
Exceptions are matched in the order in which catch clauses are listed. So if a base class clause occurs before a derived class clause, it will always match base as well as derived class exceptions. Hence, `unknown_error` object thrown, matches `cust_error::error&` at `catch` block at `LINE-2` and `cust_error::unknown_error&` at `catch` block at `LINE-3`. As the `catch` block at `LINE-2` appear first, then the output is `error`.

Question 3

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

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

int main() {
    try {
        throw "s";
    }
    catch (int x) {
        cout << "Caught 1 " << x;
    }
    catch (char x) {
        cout << "Caught 2 " << x;
    }
    catch (string x) {
        cout << "Caught 3 " << x;
    }
    catch (...) {
        cout << "Default Exception";
    }
    return 0;
}
```

What will be the output?

a) `Caught 1`

b) `Caught 2`

c) `Caught 3`

d) `Default Exception`

Answer: d)
Explanation:
No catch argument type matches the type of the thrown object. If the ellipsis `(...)` is used as the parameter of catch, then that handler can catch any exception no matter what the type of the exception thrown. This can be used as a default handler that catches all exceptions not caught by other handlers.
In the given program, the thrown exception is of type `char const*`, which does not have any match. Hence it will be caught by `catch (...)`. Hence, the correct option is d).

Question 4

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

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

template<class T> T GetMax(T& a, T& b) {    // LINE-1
    return ((a>b) ? a : b);
}

int main() {
    int i = 5, j = 6, k;
    long l = 10, m = 5, n;

    k = GetMax<int>(i, j);
    n = GetMax<long>(l, m);
    cout << k << " ";
    cout << n;

    return 0;
}
```

Fill the blank at `LINE-1`, such that the output is:
`6 10`

a) `int GetMax (int a, int b)`

b) `template <typename T> GetMax`

c) `template <typename T> T GetMax(T a, T b)`

d) `template <class T> T GetMax(T& a, T& b)`

Answer: c), d)
Explanation:
The function `GetMax` must accept parameters of int as well as long type. This can be done using template definition. By the definition of function template, `LINE-1` must be filled either as:
`template <typename T> T GetMax(T a, T b)`
or
`template <class T> T GetMax(T& a, T& b)`

Question 5

Consider the code below. *[MCQ, Marks 2]*

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

template <typename T>
T sum(T x, T y) {
    return x + y;
}

int main() {
    cout << _____________;    // LINE-1

    return 0;
}
```

What shall be the output/error when the blank space in LINE-1 is filled with the following:
(i) `sum(10, 20)`
(ii) `sum(3.14, 9.76)`
(iii) `sum(3.14, 9)`

a) Error: For all the calls, type is not instantiated

b) (i) `30`, (ii) `12.9`, (iii) `12.14`

c) (i) `20`, (ii) `12`, (iii) error: as no matching for `sum(double, int)`

d) (i) `30`, (ii) `12.9`, (iii) error: as no matching for `sum(double, int)`

Answer: d)
Explanation:
For `max(10, 20)`, the output is the sum (of `int` type) `30`.
For `max(3.14, 9.76)`, the output is the sum (of `double` type) `12.9`.
For `max(3.14, 9)`, 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 6

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

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

____________________ // LINE-1
class List {
    T arr[N];
public:
    void setVal(int x, T value) {
        arr[x] = value;
    }
    T getVal(int x) {
        return arr[x];
    }
};

int main() {
    List<int, 5> myints;
    List <double, 5> mydoubles;

    myints.setVal(3, 10);
    mydoubles.setVal(1, 3.14);
    cout << myints.getVal(3) << " ";
    cout << mydoubles.getVal(1) << " ";

    return 0;
}
```

Fill in the blank at LINE-1 such that the output is:
`10 3.14`

a) `template <class T>`

b) `template <typename T, int N = 0>`

c) `template <class T, class N = 0>`

d) `template <class T, int N>`

Answer: b), d)
Explanation :
A template must be declared with one typed parameter and one non-typed parameter as follows:
`template <typename T, int N = 0>`
or
`template <class T, int N>`

Question 7

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

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

template <class T, int i>
void repeat(T val) {
    i = 5;
    for (int j = 0; j < i; j++)
        cout << val << " ";

    return;
}

int main() {
    repeat<int, 10>(10);

    return 0;
}
```

What will be the output / error?

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

b) `10 10 10 10 10`

c) `10 10 10 10 10 0 0 0 0 0`

d) `Compiler error:  l-value required`

Answer: d)
Explanation:
Compiler error in line `i = 5;` as non-type parameters must be constant and cannot be modified. Hence, it gives compiler error.

Question 8

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

```
#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
_____________ {     // LINE-2: Fill with the correct Template signature
    return n1 + n2;
}

int main() {
    Adder<int> obj1(10, 20);
    Adder<double> obj2(3.14, 8.6);

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

    return 0;
}
```

Fill in the blanks at LINE-1 and LINE-2 with appropriate options such that the output is: `30 11.74`

a) `LINE-1: template<class T>, LINE-2: T Adder<>::Add()`

b) `LINE-1: template<class T>, LINE-2: T Adder<T>::Add()`

c) `LINE-1: template<typename T>, LINE-2: T Adder::Add()`

d) `LINE-1: template<typename T>, LINE-2: T Adder<typename T>::Add()`

Answer: b)
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 9

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

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

typedef struct complex_num {
    int r, i;
}COMPLEX;

template<class T>
T operator+(T& a, T& b) {
    return a + b;
}

__________                  // LINE-1
COMPLEX operator+(COMPLEX& a, COMPLEX& b) {
    COMPLEX c;
    c.r = a.r + b.r;
    c.i = a.i + b.i;
    return c;
}

int main() {
    int a = 10, b = 20;
    COMPLEX c1 = { 10, 20 };
    COMPLEX c2 = { 30, 40 };

    int c = a + b;
    cout << c << endl;

    COMPLEX c3 = c1 + c2;
    cout << c3.r << " , " << c3.i;

    return 0;
}
```

Fill in the blank at LINE-1 with appropriate option such that the output is:

```
30
40 , 60
```

a) `LINE-1: template<>`

b) `LINE-1: template<COMPLEX>`

c) `LINE-1: template`

d) `LINE-1: template<T>`

Answer: a)
Explanation:
The template function `operator+()` having two versions. The generic version and, a specialized version for `COMPLEX` type. For specialized type LINE-1 need to be filled as:
`template<>`.

Programming Questions

Question 1

Consider the following program. Fill in the blanks: at `LINE-1` with appropriate template header, at `LINE-2` with proper parameter-list for function `Swap()`, and at `LINE-3` with proper declaration of function variable, such that it satisfies the given test cases.

Marks: 3

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

// Write the Swap function here
____________________   // LINE-1
void Swap(_________) { // LINE-2
    __________         // LINE-3
    t = x;
    x = y;
    y = t;
}

int main() {
    int a, b;
    double s, t;
    string mr, ms;

    cin >> a >> b;
    cin >> s >> t;
    cin >> mr >> ms;

    Swap(a, b);
    Swap(s, t);
    Swap(mr, ms);

    cout << a << " " << b << " ";
    cout << s << " " << t << " ";
    cout << mr << " " << ms;

    return 0;
}
```

Public 1

```
Input:
10 20
3.14 5.10
aaa bbb
Output: 20 10 5.1 3.14 bbb aaa
```

Public 2

```
Input:
-12 34
23.4 -12.33
abc xyz
Output: 34 -12 -12.33 23.4 xyz abc
```

Private

```
Input:
100 200
-3.13 34.33
hello world
Output: 200 100 34.33 -3.13 world hello
```

Answer:

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

or

```
LINE-1:  template <typename T>
LINE-2:  T& x, T& y
LINE-3:  T t;
```

Explanation:

`LINE-1` must be template-header, hence it is either

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

or

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

`LINE-2` must be filled with template-typed parameters, hence the function header is as follows:
`void Swap(T& x, T& y)`
`LINE-3` must be having the declaration of temporary variable `t`, which will be `T t;`.

Question 2

Consider the following program. Fill in the blanks in `LINE-1` for template declaration for class A such that it satisfies the given test cases. *Marks: 3*

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

// Write the class header here
____________________    // LINE-1
class A {
public:
    T x;
    U y;
    A(T x, U y){ cout << x << ' ' << y << endl; };
};

int main() {
    int num;
    char c;

    cin >> num;
    cin >> c;

    A<int, char> a1(num, c);
    A<int> a2(num, c);

    return 0;
}
```

Public 1

```
Input: 100 a
Output:
100 a
100 97
```

Public 2

```
Input: 88 A
Output:
88 A
88 65
```

Private

```
Input: 10 x
Output:
10 x
10 120
```

Answer:

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

Explanation:

As in the statement `A<int> a2(num, c);`, both of the template type-arguments are not specified, the second argument must have a default type. Hence, the `LINE-1` must be filled by:
`template <class T, class U = int>`
as for the statement `A<int> a2(num, c);`, `c` is converted to `int`.

Question 3

Consider the following program. Class `mytype` must be a generic type. So fill in the blank at `LINE-1` with the proper `template` declaration. Fill in the blank at `LINE-2` with appropriate header for the function to add two `mytype` objects, and also fill in the blank at `LINE-3` with appropriate return statement such that it satisfies the given test cases. *Marks: 3*

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

____________________    // LINE-1
class mytype {
    T a, b;
public:
    mytype(T _a, T _b) : a(_a), b(_b) { }
    ____________________ { // LINE-2
        return mytype(____________________); // LINE-3
    }
    void show() {
        cout << a << ", " << b << endl;
    }
};

int main() {
    int i1, i2, i3, i4;
    cin >> i1 >> i2 >> i3 >> i4;

    mytype<> obj1(i1, i2);
    mytype<> obj2(i3, i4);
    mytype<> obj3 = obj1 + obj2;

    double d1, d2, d3, d4;
    cin >> d1 >> d2 >> d3 >> d4;

    mytype<double> obj4(d1, d2);
    mytype<double> obj5(d3, d4);
    mytype<double> obj6 = obj4 + obj5;

    obj3.show();
    obj6.show();

    return 0;
}
```

Public 1

```
Input:
10 20 30 40 12.03 60.01 23.88 5.14
Output:
40, 60
35.91, 65.15
```

Public 2

```
Input:
67 4 100 75
6.09 6.12 90.5 87.5
Output:
167, 79
96.59, 93.62
```

Private

```
Input:
7 900 54 300
7.85 9.12 900.01 6.55
Output:
61, 1200
907.86, 15.67
```

Answer:

```
LINE-1:  template<class T = int>
```

or

```
LINE-1:  template<typename T = int>
LINE-2:  mytype operator+(mytype x)
LINE-3:  a + x.a, b + x.b
```

or

```
LINE-3:  this->a + x.a, this->b + x.b
```

Explanation:

At LINE-1, the template type need to be created. As per the test cased the default tupe must be `int`, so LINE-1 must be:

```
LINE-1:  template<class T = int>
```

or

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

At LINE-2, the header of the function must be

```
LINE-2:  mytype operator+(mytype x)
```

At LINE-3, the sum must be calculated either by

```
LINE-3:  a + x.a, b + x.b
```

or

```
LINE-3:  this->a + x.a, this->b + x.b.
```

Question 4

Consider the following program. Fill in the blanks at `LINE-1` to create user-defined exception `DivideByZeroException`. Fill in the banks at `LINE-2` amd LINE-3 to throw the appropriate exceptions so that it satisfies the given test cases. *Marks: 3*

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

// declare a user-defined exception
class DivideByZeroException : ____________________ { // LINE-1
public:
    virtual const char* what() const throw();
};

void divide(int i, int j) {
    try {
        if (j == 0)
            // throw the exception object
            ____________________; // LINE-2
        else if (i == 0)
            ____________________; // LINE-3
        else
            cout << (double)i / j;
    }
    catch (int& e) {
        cout << "result is always: " << e;
    }
    catch (DivideByZeroException e) {
        cout << e.what() << endl;
    }
}

const char* DivideByZeroException::what() const throw() {
    return "cannot divide by 0";
}

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

    divide(i, j);

    return 0;
}
```

Public 1

```
Input:  10 0
Output:  cannot divide by 0
```

Public 2

```
Input:  0 20
Output:  result is always:  0
```

Public 3

```
Input:  40 5
Output:  8
```

Private

```
Input:  100 0
Output:  cannot divide by 0
```

Answer:

`LINE-1:  exception`
OR `public exception`
OR `protected exception`
OR `private exception`
`LINE-2:  throw DivideByZeroException()`
`LINE-3:  throw 0`
Explanation:
`DivideByZeroException` is a user-defined exception, hence it needs to be inherited from class `exception`.
At LINE-2, fill in blank to throw the user-exception `DivideByZeroException` as:
`throw DivideByZeroException()`
At LINE-3, fill in blank to throw the exception as follow:
`LINE-3:  throw 0`.