Programming in Modern C++: Assignment Week 8

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 following program. *[MCQ, Marks 2]*

```
#include <iostream>
using namespace std;
void testIt(int i) {
    if (i)
        throw i;
    else
        throw "zero";
}
int main() {
    try {
        // statement-1
    }
    catch (int e) {
        cout << "int" << endl;
    }
    catch (float e){
        cout << "float" << endl;
    }
    catch (double e){
        cout << "duoble" << endl;
    }
    catch (const char* e) {
        cout << "string" << endl;
    }
    catch (...) {
        cout << "anything else" << endl;
    }
    return 0;
}
```

What will be the outputs in consecutive two runs if `statement-1` is replaced by (i) `testIt(3.14);` and (ii)`testIt(0);` respectively?

a) (i) `int` and (ii) `string`

b) (i) `double` and (ii) `string`

c) (i) `float` and (ii) `anything else`

d) (i) `double` and (ii) `anything else`

Answer: a)
Explanation:
For the call `testIt(3.14)`, the `double` value is type cast to `int`. Thus, when the exception of `int` type is forwarded to the `main`, it would be caught by `catch(int i){ ... }`.
For the call `testIt(0)`, the exception type is `const char*` type. Thus, when the exception is forwarded to the `main`, it would be caught by `catch(const char* e){ ... }`.

Question 2

Consider the following code segment. *[MCQ, Marks 2]*

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

class InternalError {
    public:
        virtual const char* what() const throw() {
            return "internal exception";
        }
};
class DBError : public InternalError {
    public:
        virtual const char* what() const throw() {
            return "DB exception";
        }
};
class SQLError : public DBError {
    public:
        virtual const char* what() const throw() {
            return "SQL exception";
        }
};

int main(){
    try{
        throw SQLError();
    }
    catch(InternalError& e){ cout << e.what(); }
    catch(DBError& e){ cout << e.what(); }
    catch(SQLError& e){ cout << e.what(); }
    catch(...){ cout << "default" << endl; }
    return 0;
}
```

What will be the output?

a) `internal exception`

b) `SQL exception`

c) `DB exception`

d) `default`

Answer: b)
Explanation:
The `proc` function throws `ServerExceptions::SQLError` exception which will be caught, but rethorws `ServerExceptions::DBError` exception. The exception will be forwarded to `main`. Since the first catch block handles `ServerExceptions::InternalError` exception which is of parent type of the exception thrown, the exception gets handed in the same block. However, since `e.what()` is a polymorphic call (object type is `SQLError`) it prints `SQL exception`.

Question 3

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
using namespace std;
namespace ServerExceptions{
    class InternalError {};
    class DBError : public InternalError {};
    class SQLError : public DBError {};
};

void proc(){
    try {
        throw ServerExceptions::SQLError();
    } catch(ServerExceptions::SQLError& e){
        throw ServerExceptions::DBError();
    }
}


int main() {
    try{
        proc();
    }
    catch (ServerExceptions::InternalError&) {  //LINE-1
        cout << "internal error" << endl;
    }
    catch (ServerExceptions::SQLError&) {       //LINE-2
        cout << "sql error" << endl;
    }
    catch (ServerExceptions::DBError&) {        //LINE-3
        cout << "db error" << endl;
    }
    catch (...) {                               //LINE-4
        cout << "default" << endl;
    }
    return 0;
}
```

What will be the output?

a) `internal error`

b) `sql error`

c) `db error`

d) `default`

Answer: a)
Explanation:
The `proc` function throws `ServerExceptions::SQLError` exception which will be caught, but rethorws `ServerExceptions::DBError` exception. The exception will be forwarded to `main`. Since the first catch block handles `ServerExceptions::InternalError` exception which is of parent type of the exception thrown, the output becomes `internal error`.

Question 4

Consider the following code segment. *[MCQ, Marks 2]*

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

void throwIt(){
    try{
        throw "divide by zero";
    }
    catch(int&){ throw; }
}

int main(){
    try{
        throwIt();
    }
    catch(...){ cout << "default"; }    //LINE-1
    catch(int& e){ cout << "int"; }
    catch(double& e){ cout << "double"; }
    return 0;
}
```

What will be the output/error?

a) `default`

b) `int`

c) `double`

d) `Compiler error LINE-1: '...' handler must be the last handler for its try block`

Answer: d)
Explanation:
`catch(...) { ... }` must be the last handler for its try block. Thus, it generates an error at `LINE-1`.

Question 5

Consider the following code segment. *[MSQ, Marks 2]*

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

template<class T>
class Modifier{
    T val;
    public:
        Modifier(T _val = 0) : val(_val) { }
        T subtract(int d){
            T t = val - d;
            return t;
        }
};

___________________________    //LINE-1
__________________________ {   //LINE-2
    char* val;
    public:
        Modifier(const char* _val = 0) : val(strdup(_val)) { }
        char* subtract(int d){
            char* buf = (char*)malloc(strlen(val) - d + 1);
            int i;
            for(i = 0; i < strlen(val) - d; i++)
                buf[i] = val[i];
            buf[i] = '\0';
            return buf;
        }
};

int main(){
    Modifier<double> d = 26.44;
    Modifier<const char*> s("programming");
    cout << d.subtract(4) << ", ";
    cout << s.subtract(4);
    return 0;
}
```

Identify the appropriate option to fill in the blanks at `LINE-1` and `LINE-2` such that the output becomes `22.44, program`.

a) `LINE-1: template<T>`
   `LINE-2: class Modifier<char*>`

b) `LINE-1: template<>`
   `LINE-2: class Modifier<char*>`

c) `LINE-1: template<>`
   `LINE-2: class Modifier<const char*>`

d) `LINE-1: template<const char*>`
   `LINE-2: class Modifier`

Answer: c)
Explanation:
To extend the template for specialized type `const char*`, the `LINE-1` and `LINE-2` must be filled up as follows:

```
LINE-1:  template<>
LINE-2:  class Modifier<const char*>
```

It is kept MSQ intensionally.

Question 6

Consider the following code segment. *[MCQ, Marks 2]*

```
#include <iostream>
#include <string>
using namespace std;
template <typename T, int n>            //LINE-1
class MultiPrint{
    T val;
    public:
        MultiPrint(T _val) : val(_val){}
        void output(){
            for(int i = 0; i < n; i++)
                cout << val << " ";
        }
};
int main(){
    int n = 3;
    MultiPrint<string, n> d("hello");    //LINE-2
    d.output();
    return 0;
}
```

What will be the output / error?

a) `3`

b) `hello hello hello`

c) `Compiler error at LINE-1:  non-type argument is not allowed`

d) `Compiler error at LINE-2:  non-type argument must be constant`

Answer: d)
Explanation:
In C++ template, non-type argument n in template must be a constant.

Question 7

Consider the following function template. *[MSQ, Marks 2]*

```
template<typename T>
void print(T x, T y){
    cout << x << " " << y;
}
```

From among the following option which generate/s compiler error?

a) `print(10, 'a');`

b) `print(10, 30);`

c) `print(10.34, 30);`

d) `print((int)'a', 30)`

Answer: a), c)
Explanation:
For function `print`, both the arguments must be of the same type, otherwise it generates a compiler error. Thus, the calls in options a) and c) generate compiler errors.

Question 8

Consider the following class definition in C++. *[MCQ, Marks 2]*

```
class Computation{
    public:
        int sum(int a, int b) { return a + b; }
};
```

Identify the appropriate function pointer declaration that can point to the function `sum` belongs to the class `Computation` as `fpm = &Computation::sum;`.

a) `typedef int (Computation::*fpm) (int, int);`

b) `int *Computation::fpm(int, int);`

c) `int (*Computation::fpm)(int, int);`

d) `int (Computation::*fpm)(int, int);`

Answer: d)
Explanation:
The appropriate syntax to declaration a function pointer to the member function `sum` of class `Computation` is option d).

Question 9

Consider the following code segment. *[MCQ, Marks 2]*

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

struct Division {
    int r;
    Division(int _r = 0) : r(_r) { }
    ___________________________ {    //LINE-1
        int d = d1 / d2;
        r = d1 % d2;
        return d;
    }
};

int main(){
    int a = 35, b = 3;
    Division div(0);
    int d = div(a, b);
    cout << "d = " << d << endl;
    cout << "r = " << div.r << endl;
    return 0;
}
```

Identify the appropriate option to fill in the blank at `LINE-1` such that the output becomes

```
d = 11
r = 2
```

a) `int operator(int d1, int d2)`

b) `int operator()(int d1, int d2)`

c) `void operator()(int d1, int d2)`

d) `int operator(int d1, int d2)()`

Answer: b)
Explanation:
The appropriate header to overload the function operator in this case is option b).

Programming Questions

Question 1

Consider the program below.

- Fill in the blank at `LINE-1` with template declaration.
- Fill in the blank at `LINE-2` to define the header of function `extract(.)`.

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

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

_____________________    //LINE-1
class Filter{
    T ub, lb;
    public:
        Filter(T _lb = 0, T _ub = 0) : ub(_ub), lb(_lb) { }
        vector<T> extract(vector<T> tVec);
};
____________________________________________________ {    //LINE-2
    vector<T> rVec;
    for(int i = 0; i < tVec.size(); i++){
        if(tVec[i] <= ub && tVec[i] >= lb)
            rVec.push_back(tVec[i]);
    }
    return rVec;
}

int main(){
    int i1, i2;
    char d1, d2;
    cin >> i1 >> i2 >> d1 >> d2;
    Filter<int> flt1(i1, i2);
    Filter<char> flt2(d1, d2);
    int arr1[] = {30, 10, 50, 60, 80, 20, 40, 70, 90};
    char arr2[] = {'a', 'n', 'i', 'm', 'l', 's'};
    vector<int> iVec(arr1, arr1 + 9);
    vector<char> cVec(arr2, arr2 + 6);
    vector<int> rVec1 = flt1.extract(iVec);
    vector<char> rVec2 = flt2.extract(cVec);
    for(int i = 0; i < rVec1.size(); i++)
        cout << rVec1[i] << ", ";
    cout << endl;
    for(int i = 0; i < rVec2.size(); i++)
        cout << rVec2[i] << ", ";
    return 0;
}
```

Public 1

```
Input: 30 50 f x
```

```
Output:
30, 50, 40,
n, i, m, l, s,
```

Public 2

```
Input: 30 60 m s
Output:
30, 50, 60, 40,
n, m, s,
```

Private

```
Input: 30 50 l i
Output:
30, 50, 40,
```

Answer:
`LINE-1:  template<class T>`
or
`LINE-1:  template<typename T>`
`LINE-2:  template<class T> vector<T> Filter<T>::extract(vector<T> tVec)`
Explanation:
At `LINE-1`, the class template can be declared as:
`template<class T>`
or
`template<typename T>`.
At `LINE-2`, the header for the function `extract` belong to `Filter` class can be defined as:
`template<class T> vector<T> Filter<T>::extract(vector<T> tVec)`

Question 2

Consider the following program.

- Fill in the blank at `LINE-1` with the appropriate inheritance statement.
- Fill in the blank at `LINE-2` with the appropriate return statement.
- Fill in the blank at `LINE-3` with the appropriate declaration of generic array `items`.
- Fill in the blank at `LINE-4` with the appropriate function header to overload `operator[]`.
- Fill in the blank at `LINE-5` to throw exception `InvalidAccess`.

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

```
#include<iostream>
#include<exception>
#include<vector>
#include<cstdlib>
using namespace std;
class InvalidAccess : ________________ { // LINE-1
    public:
        virtual const char* what() const throw() {
            ________________________;    //LINE-2
        }
};
template<class T>
class ItemList{
    int n;
    _________________;    //LINE-3
    public:
        ItemList(vector<T> _items) : n(_items.size()),
                        items((T*)malloc(n * sizeof(int))){
            for(int i = 0; i < n; i++){
                items[i] = _items[i];
            }
        }

        _________________________{     //LINE-4
            if(i >= 0 && i < n){
                return items[i];
            }
            ____________________;      //LINE-5
        }
};

int main(){
    int idx[3];
    for(int i = 0; i < 3; i++){
        cin >> idx[i];
    }

    vector<int> iVec;
    vector<char> cVec;
    for(int i = 0; i < 5; i++){
        iVec.push_back((i + 1) * 10);
        cVec.push_back(65 + i);
    }

    ItemList<int> iList(iVec);
    ItemList<char> cList(cVec);

    for(int i = 0; i < 3; i++){
        try{
            cout << iList[idx[i]] << ", ";
            cout << cList[idx[i]] << endl;
        }catch(InvalidAccess e){
            cout << e.what() << endl;
        }
    }
    return 0;
}
```

Public 1

```
Input: 2 -3 6
Output:
30, C
invalid access
invalid access
```

Public 2

```
Input: 2 4 6
Output:
30, C
50, E
invalid access
```

Private

```
Input: 9 1 0
Output:
invalid access
20, B
10, A
```

Answer:
```
LINE-1:  public exception
LINE-2:  return "invalid access"
LINE-3:  T* items
LINE-4:  T& operator[](int i)
LINE-5:  throw InvalidAccess()
```
Explanation:
At `LINE-1`, the exception `InvalidAcess` must inherit `exception` class as:
`class InvalidAccess :  public exception`.
At `LINE-2`, the return statement must be:

`return "invalid access";`
At `LINE-3`, the generic array must be declared as:
`T* items;`
At `LINE-4`, the function header to overload `operator[]` must be:
`T& operator[](int i)`
At `LINE-5`, the exception must be thrown as:
`throw InvalidAccess();`

Question 3

Consider the following program that consider a list of strings as input and prints them in sorted order of their length.

- Fill in the blanks at `LINE-1` with the function header to overload the function operator.
- Fill in the blanks at `LINE-2` with an appropriate return statement such that it sorts them according to the length of the strings (in case, if the two strings have the same length then it maintains the actual order).

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

```
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
using namespace std;
struct less_mag {
    _________________________________ {    //LINE-1
        _____________________________ ;    //LINE-2
    }
};

int main(){
    vector<string> iVec;
    int n;
    string na;
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> na;
        iVec.push_back(na);
    }
    sort(iVec.begin(), iVec.end(), less_mag());
    for(int i = 0; i < iVec.size(); i++)
        cout << iVec[i] << " ";
    return 0;
}
```

Public 1

```
Input: 4 Ducks Chicken Dove Pig
Output: Pig Dove Ducks Chicken
```

Public 2

```
Input: 5 grapes figs jackfruit apple blueberries
Output: figs apple grapes jackfruit blueberries
```

Private

```
Input: 7 Cow Sheep Dog Goat Horse Mouse Ox
Output: Ox Cow Dog Goat Sheep Horse Mouse
```

Answer:
`LINE-1:  bool operator()(string x, string y)`
`LINE-2:  return (x.length() < y.length())`
Explanation:
At `LINE-1`, the function header to overload the function operator is:
`bool operator()(string x, string y)`
At `LINE-2`, the return statement for sorting the elements by the length of the strings:
`return (x.length() < y.length())`