Programming in Modern C++: Assignment Week 9

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<cstdio>
using namespace std;

int main(){
    int a = 0x067, b = 067, c = 67;
    char d = 67;    //LINE-1
    printf("%d %d %d %c", a, b, c, d);
    return 0;
}
```

What will be the output/error?

a) `67 67 67 67`

b) `67 67 67 C`

c) `103 55 67 C`

d) `Compiler error at LINE-1:  type cast from int -> char is invalid`

Answer: c)
Explanation:
The integer `a = 0x067` is assigned to a hexadecimal value. The corresponding decimal value is `103`.
The integer `b = 067` is assigned to an octal value. The corresponding decimal value is `55`.
The integer `c = 67` is assigned to a decimal value.
The character `d = 67` is assigned to an integer value, which would the considered as ASCII code of `'C'`.
Thus the output is `103 55 67 C`.

Question 2

Match the appropriate descriptions about the `fseek` function calls.
Here, `fp` is function pointer. *[MCQ, Marks 2]*

Function call	**Description**
1. `fseek(fp, 10, SEEK_SET)`	A. Move the file pointer to the end of the file
2. `fseek(fp, -10, SEEK_CUR)`	B. Move the file pointer forward from the beginning of the file by 10 positions
3. `fseek(fp, 0, SEEK_END)`	C. Move the file pointer backwards from the current position in the file by 10 positions
4. `fseek(fp, -10, SEEK_END)`	D. Move the file pointer backwards from the end of the file by 10 positions

a) `1-A, 2-D, 3-C, 4-B`

b) `1-A, 2-C, 3-D, 4-B`

c) `1-B, 2-A, 3-B, 4-D`

d) `1-B, 2-C, 3-A, 4-D`

Answer: d)
Explanation:

- `fseek(fp, 10, SEEK_SET)` move the file pointer forward from the beginning of the file by 10 positions.
- `fseek(fp, -10, SEEK_CUR)` moves the file pointer backwards from the current position in the file by 10 positions
- `fseek(fp, 0, SEEK_END)` moves the file pointer to the end of the file
- `fseek(fp, -10, SEEK_END)` moves the file pointer backwards from the end of the file by 10 positions

Question 3

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

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

int main(){
    FILE *infp, *outfp;
    if((infp = fopen("myfile.txt", "r")) == NULL)
        return -1;
    if((outfp = fopen("procfile.txt", "w")) == NULL)
        return 2;
    int c;
    while((c = fgetc(infp)) != EOF)
        if(c == ' ' || c == '\n');      //LINE-1
        else
            fputc(c, outfp);
    fclose(infp);
    fclose(outfp);
    return 0;
}
```

Choose the correct option regarding the program.

a) It makes the exact copy of the contents from the file `myfile.txt` to the file `procfile.txt`

b) It makes the exact copy of the contents from the file `procfile.txt` to the file `myfile.txt`

c) It makes the exact copy of the contents from the file `myfile.txt` to the file `procfile.txt` without spaces and newlines.

d) It generates compiler error at `LINE-1` since the `;` is placed at wrong position

Answer: c)
Explanation:
The program copies every character from the file `myfile.txt` to the file `procfile.txt` except the spaces and newline characters.

Question 4

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

```
#include <iostream>
#include <iomanip>

int main () {
    std::cout.setf(std::ios::showpoint);
    std::cout << std::setfill ('0') << std::setw (10) << 11.0;
    return 0;
}
```

What will be the output?

a) `00000011.0`

b) `0000000011`

c) `00011.0000`

d) `0000000011.00`

Answer: c)
Explanation:
The statement `std::cout.setf(std::ios::showpoint);` prints the decimal point with four 0s by default (which has a width 5).
The statement `std::setw (10)` sets the width to 10, where 5 + 2 (width of 11) = 7 positions are already used.
The statement `std::setfill ('0')` makes the rest 3 positions to be filled with 0s.

Question 5

Consider the file `myfile.txt` has a single line as follows:
`pointer to the array where the read objects are stored`
Consider the following code segment. *[MSQ, Marks 2]*

```
#include <iostream>
#include <fstream>
using namespace std;
int main () {
    ifstream myfile("myfile.txt");
    char c;
    int i = 0;
    if (myfile.is_open()) {
        while (!myfile.eof()) {
            _________________;    //LINE-1
            if(c == ' ')
                i++;
        }
        myfile.close();
        cout << i;
    }
    else
        cout << "Unable to open file";
}
```

Identify the appropriate option to fill in the blank at `LINE-1` such that the program prints the number of words in the file `myfile.txt` if the file exists. Otherwise, prints `Unable to open file`. Here, each word is separated by a single space.

a) `getline(myfile, c)`

b) `c = myfile.get()`

c) `myfile >> c`

d) `myfile.get(c)`

Answer: b), d)
Explanation:
The statement `getline(myfile, c)` reads a line from the given fine, but `c` is a `char`. Thus it generates compiler error.
The statement `myfile >> c` reads a char, but skips spaces and newlines. Thus it generates wrong word count.
The statements `c = myfile.get()` and `myfile.get(c)`, both reads a character from the file, so prints the number words properly.

Question 6

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

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

template<class Itr, class T>
void MinMax(____________________________________) {    //LINE-1
    max = *++first;
    min = *first;
    while (first != last) {
        if(*first > max)
            max = *first;
        else if(*first < min)
            min = *first;
        ++first;
    }
}

int main(){
    int min = 0, max = 0;
    int iArr[] = {5, 6, 7, 1, 2, 9, 3, 4};
    MinMax(iArr, iArr + sizeof(iArr) / sizeof(*iArr), max, min);
    cout << min << ", " << max;
    return 0;
}
```

Fill in the blank at `LINE-1` such that the program will print `1, 9`.

a) `Itr first, Itr last, T max, T min`

b) `Itr first, Itr last, T& max, T& min`

c) `T first, T last, Itr& max, Itr& min`

d) `T first, T last, T& max, T& min`

Answer: b), c)
Explanation:
Since the first two arguments are of the same type, which is `int*`, and the next two arguments are of the same type that is `int` along with passed-by-reference, the options b) and c) are correct.

Question 7

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

```
#include <iostream>
#include <list>
#include <numeric>
#include <functional>
using namespace std;

double compute(list<int>& li) {
    double result = accumulate(_________________________________);    //LINE-1
    return result;
}
int main() {
    int arr[] = { 10, 20, 30, 40 };
    list<int> li(arr, arr + sizeof(arr) / sizeof(*arr));
    cout << compute(li) << endl;
    return 0;
}
```

Identify the appropriate option such that it multiplies the elements of list `li` and then divide it 2. In this case, the program prints `120000` which is computed as:

$$\frac{10 \times 20 \times 30 \times 40}{2} = \frac{240000}{2} = 120000.$$

a) `li.begin(), li.end(), 1, multiplies<double>()`

b) `li.begin(), li.end(), 0.5, multiplies<double>()`

c) `li.begin(), li.end(), 0.5, multiplies<int>()`

d) `li.begin(), li.end(), 0, multiplies<int>()`

Answer: b)
Explanation:
Since the result has to the half of the product of the elements list `li` the initial value to be multiplied with will be `0.5`.
If we consider `multiplies<int>()`, the initial value `0.5` will become `0`. Thus, the entire result becomes `0`. Thus, it shall be `multiplies<double>()`.

Question 8

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

```
#include <iostream>
#include <list>
#include <algorithm>
#include <numeric>
using namespace std;

struct operation1{
    int operator()(int i, int j){ return i + j; }
};

int operation2(int i, int j){ return i * j; }

int main() {
    list<int> li1  { 1, 2, 3 };
    list<int> li2  { 30, 20, 10 };

    int result = ___________________________________________;    //LINE-1
    cout << result;
    return 0;
}
```

Identify the appropriate call to `inner_product` function to fill in the blank at `LINE-1` such that it prints `100` as output.

a) `inner_product(li1.begin(), li1.end(), li2.begin(), 0, operation1(), operation2)`

b) `inner_product(li1.begin(), li1.end(), li2.begin(), 0, operation1, operation2())`

c) `inner_product(li1.begin(), li1.end(), li2.begin(), 1, operation1(), operation2)`

d) `inner_product(li1.begin(), li1.end(), li2.end(), 1, operation1(), operation2)`

Answer: a)
Explanation:
The code by `inner_product` function is:

```
template<class In, class In2, class T, class BinOp, class BinOp2 >
T inner_product(In first, In last, In2 first2, T init, BinOp op, BinOp2 op2) {
    while(first!=last) {
        init = op(init, op2(*first, *first2));
        ++first; ++first2;
    }
    return init;
}
```

Thus, a) is the correct option.

Question 9

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

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

int main() {
    char cArr[] = { 'w', 'o', 'r', 'l', 'd' };
    int l = sizeof(cArr) / sizeof(*cArr);
    vector<char> cVec(l);

    ____________________________________;    //LINE-1

    for(vector<char>::iterator it = cVec.begin(); it != cVec.end(); ++it)
        cout << *it;
    return 0;
}
```

Identify the appropriate call to `copy` function to fill in the blank at `LINE-1` such that it prints `world` as output.

a) `copy(cArr, cArr.end(), cVec.begin())`

b) `copy(&cArr[0], &cArr[l], cVec.begin())`

c) `copy(cArr, cArr + l, cVec.begin())`

d) `copy(cVec.begin(), cVec.end(), cArr)`

Answer: b), c)
Explanation:
The syntax of `copy` function is as follows:

```
template<class InputIterator, class OutputIterator>
OutputIterator copy(InputIterator first, InputIterator last, OutputIterator result)
```

The correct options are – b) and c).

Programming Questions

Question 1

Consider the program below that merges two orders into a final order and prints it.

- Fill in the blank at `LINE-1` with appropriate statement to add the items with quantity in order `od1` to `final_od`.
- Fill in the blank at `LINE-2` with appropriate statement to add the items with quantity in order `od2` to `final_od`. Add the quantity if the item already exists in `final_od`.

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

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

map<string, int> merge_order(map<string, int> od1, map<string, int> od2){
    map<string, int> final_od;
    for (map<string, int>::iterator it = od1.begin(); it != od1.end(); ++it)
        ______________________________________;    //LINE-1
    for (map<string, int>::iterator it = od2.begin(); it != od2.end(); ++it)
        ______________________________________;    //LINE-2
    return final_od;
}

void show(map<string, int> od){
    for (map<string, int>::iterator it = od.begin(); it != od.end(); ++it)
        cout << it->first << " => " << it->second << endl;
}

int main() {
    map<string, int> order1;
    map<string, int> order2;
    string item;
    int qty;
    for(int i = 0; i < 3; i++){
        cin >> item >> qty;
        order1[item] = qty;
    }
    for(int i = 0; i < 3; i++){
        cin >> item >> qty;
        order2[item] = qty;
    }

    map<string, int> final_order = merge_order(order1, order2);
    show(final_order);
    return 0;
}
```

Public 1

Input:

```
pen 2
mobile 1
cap 4
file 2
pen 4
mobile 1
Output:
cap => 4
file => 2
mobile => 2
pen => 6
```

Public 2

```
Input:
paper 10
pen 1
book 2
paper 5
pencil 2
file 1
Output:
book => 2
file => 1
paper => 15
pen => 1
pencil => 2
```

Private

```
Input:
apple 5
orange 5
mango 3
mango 7
apple 5
orange 5
Output:
apple => 10
mango => 10
orange => 10
```

Answer:

```
LINE-1:  final_od[it->first] = od1[it->first]
LINE-2:  final_od[it->first] += od2[it->first]
```

Explanation:

At `LINE-1`, we have to add the items and their quantity from order `od1` to `final_order` as:
`final_od[it->first] = od1[it->first]`

At `LINE-2`, we have to add the items and their quantity from order `od2` to `final_order` (add the quantity if the item already exists) as:
`final_od[it->first] += od2[it->first]`

Question 2

Consider the following program which considers the following as input – the number of employees, name and salary of each employee, followed by the type of sorting (1 for sort by name and 2 for sort by salary). It prints the employees' information in the corresponding sorted order. Complete the program with the following instructions.

- Fill the missing code segments at `code-segment-1` and `code-segment-2` with the appropriate overriding of function operator.

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

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

class employee{
    private:
        string name;
        double salary;
    public:
        employee(string _name, double _salary) : name(_name), salary(_salary){}
        string getName() const{
            return name;
        }
        double getSalary() const{
            return salary;
        }
        friend ostream& operator<<(ostream& os, const employee& e);
};

ostream& operator<<(ostream& os, const employee& e){
    os << e.name << '-' << e.salary << endl;
    return os;
}

struct compareByName{
    //code-segment-1
};

struct compareBySalary{
    //code-segment-2
};

int main() {
    int n;              //number of employees
    cin >> n;
    vector<employee> emps;
    for(int i = 0; i < n; i++){
        string na;
        double sa;
```

```
        cin >> na >> sa;    //employee's name and salary
        employee e(na, sa);
        emps.push_back(e);
    }
    int t;              //sort option
    cin >> t;
    if(t == 1)
        sort(emps.begin(), emps.end(), compareByName());
    else if(t == 2)
        sort(emps.begin(), emps.end(), compareBySalary());
    for(vector<employee>::iterator p = emps.begin(); p != emps.end(); p++){
        cout << *p;
    }
    return 0;
}
```

Public 1

```
Input:
4
deep 20000
riya 15000
vinay 18000
nilima 22000
2
Output:
riya-15000
vinay-18000
deep-20000
nilima-22000
```

Public 2

```
Input:
5
kamal 23000
rahul 18000
vinita 20000
anuska 17000
rabin 15000
1
Output:
anuska-17000
kamal-23000
rabin-15000
rahul-18000
vinita-20000
```

Private

```
Input:
3
sandesh 30000
```

```
ram 10000
viky 20000
2
Output:
ram-10000
viky-20000
sandesh-30000
```

Answer:

```
code-segment-1:
bool operator()(const employee& e1, const employee& e2) const {
    return e1.getName() < e2.getName();
}

code-segment-2:
bool operator()(const employee& e1, const employee& e2) const {
    return e1.getSalary() < e2.getSalary();
}
```

Explanation:
At `code-segment-1`, the overloading of the function operator that sorts the employees by name can be done as follows:

```
bool operator()(const employee& e1, const employee& e2) const {
    return e1.getName() < e2.getName();
}
```

At `code-segment-2`, the overloading of the function operator that sorts the employees by salary can be done as follows:

```
bool operator()(const employee& e1, const employee& e2) const {
    return e1.getSalary() < e2.getSalary();
}
```

Question 3

Consider the following program that filters the elements from a given vector `iVec` within a given upper and lower bounds and prints them.

- Fill in the blanks at `LINE-1` with appropriate template declaration.
- Fill in the blanks at `LINE-2` with an appropriate condition of the `while` loop.
- Fill in the blanks at `LINE-3` with an appropriate call to function `find_if` such that it returns an iterator object to the first element from `iVec` that match the given predicate with upper and lower bounds.

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

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

template<class T>
struct Filter{
    T ub, lb;
    Filter(T _lb = 0, T _ub = 0) : ub(_ub), lb(_lb) { }
    bool operator()(T i){ return (i <= ub && i >= lb); }
};


_________________________________     //LINE-1
T find_if(T first, T last, Pred pred) {
    while (_____________________________) ++first;    //LINE-2
        return first;
}

void printAllFiltered(vector<int> iVec, int lb, int ub){
    Filter<int> f(lb, ub);
    ________________________________________________;    //LINE-3
    while(p != iVec.end()){
        cout << *p << " ";
        p = find_if(++p, iVec.end(), f);
    }
}

int main(){
    vector<int> iVec {7, 8, 1, 4, 2, 5, 6, 3};
    int l, u;
    cin >> l >> u;
    printAllFiltered(iVec, l, u);
    return 0;
}
```

Public 1

```
Input: 5 9
Output: 7 8 5 6
```

Public 2

```
Input: 3 7
Output: 7 4 5 6 3
```

Private

```
Input: 1 10
Output: 7 8 1 4 2 5 6 3
```

Answer:

```
LINE-1:  template<class T, class Pred>
LINE-2:  first != last && !pred(*first)
LINE-3:  vector<int>::iterator p = find_if(iVec.begin(), iVec.end(), f)
```

Explanation:

At `LINE-1`, the function the appropriate funcition template is:
`template<class T, class Pred>`
At `LINE-2`, the `while` loop condition must be:
`first != last && !pred(*first)`
At `LINE-3`, the first element match the predicate can be find out as:
`vector<int>::iterator p = find_if(iVec.begin(), iVec.end(), f)`