Programming in C++: Assignment Week 2

Total Marks : 20

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

August 23, 2020

Question 1

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

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

#define X 5

int main() {
    int n = 10;
    X = n; // LINE-1

    cout << X;

    return 0;
}
```

What will be the output/error of the above code?

a) `5`

b) `10`

c) `0`

d) Compilation error at LINE-1: lvalue required as left operand of assignment.

Answer: d)
Explanation:
Manifest constants, declared as `#define` in a program, are considered as constant throughout the program. So, this rule is violated at `LINE-1` which gives compilation error.

Question 2

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

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

int main() {
    int n = 2, m = 3;
    int * const p;  // LINE-1

    p = &n;         // LINE-2
    cout << *p;

    return 0;
}
```

What will be the output of /error in the above code?

a) `2`

b) ⟨*garbage_value*⟩

c) Compilation error at LINE-1: uninitialized const 'p'.

d) Compilation error at LINE-2: assignment of read-only variable 'p'.

Answer: c),d)
Explanation:
Here, the pointer `p` is constant. So, we have to initialize the pointer while declaring. So, we will get compilation error at LINE-1. As `p` is constant pointer, the value can't be changed after declaration. So, there is compilation error at LINE-2.

Question 3

Consider below code segment. *[MSQ, Marks 2]*

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

struct complex{
    int re, im;
    void  print(){ cout << re << "+i" << im; }
};

___________________________________{            //Line-1
    struct complex c3={0,0};
    c3.re = c1.re+c2.re;
    c3.im = c1.im+c2.im;
    return c3;
}

int main(){
    struct complex c1={2,5},c2{3,-2};
    struct complex t = c1 + c2;
    t.print();
    return 0;
}
```

Complete operator overloading for structure `complex` at `Line-1` so that the output is "5+i3".

a) `complex operator+(complex &c1, complex &c2)`

b) `complex operator+(const complex &c1, const complex &c2)`

c) `operator+(complex &c1, complex &c2)`

d) `complex +(complex &c1, complex &c2)`

Answer: a), b)
Explanation:
We need to overload addition operator for the structure `complex`. It can be done as
`complex operator+(complex &c1, complex &c2)`
or,
`complex operator+(complex &c1, complex &c2)`

Question 4

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

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

int main() {
    int a = 5;
    int &b = a;

    ++a;
    ++b;

    a = a + b;
    cout << a;

    return 0;
}
```

a) 10

b) 11

c) 13

d) 14

Answer: d)
Explanation:
As `b` is a reference of `a`, both variables point to same memory location. So, `++a` and `++b` increment the initial value of `a` two times and become 7. Now, this value '7' is for both the variables `a` and `b`. So, the equation is substituted as `a = 7 + 7`. The result is 14.

Question 5

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

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

void fun(int a = 0) { cout << "1st" << endl; }

void fun() { cout << "2nd" << endl; }

int main() {
    fun(); // LINE-1

    return 0;
}
```

What will be the output/error of the above code?

a) `1st`

b) `2nd`

c) `1st`
`2nd`

d) Compilation error at LINE-1: call of overloaded `fun()` is ambiguous.

Answer: d)
Explanation:
The call of `fun()` is ambiguous due to the default parameter in the first definition and no parameter in second definition. So, it will give compilation error at LINE-1.

Question 6

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

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

int main() {
    int a = 2;
    int &ra = a;
    const int &cra = a;
    const int &cra_1 = a + 1;

    cout << (&a == &ra) << "  " << (&a == &cra) << "  " << (&a == &cra_1);

    return 0;
}
```

What will be the output of the above code?

a) 0 0 0

b) 1 1 0

c) 1 0 0

d) 1 1 1

Answer: b)
Explanation:
Since `ra` is a reference to `a` which is just an alias - not an allocation. So they must have the same address. Hence, `&a == &ra`. (a) is ruled out.

Since `cra` is a constant reference to `a` which is just an alias too (through which `a` cannot be changed, though) - it is not an allocation too. So they must have the same address. Hence, `&a == &cra`. (c) is ruled out.

Since `cra_1` is a constant reference to a new expression `a + 1`, it needs a temporary allocation to store the value of this constant expression and a new address. Hence, `&a != &cra_1`. (d) is ruled out.

Hence, (b).

Question 7

What is the output/error in the following code? *[MCQ, Marks 2]*

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

void fun(int &a, int b) {
    a = a + b;
}

int main() {
    int a = 10;

    fun(a, a);

    cout << a;

    return 0;
}
```

a) `20`

b) `10`

c) `0`

d) ⟨*garbage_value*⟩

Answer: a)
Explanation:
The variable `a` is sent to both the parameters of the function `fun()`. First parameter `a` is passed as call by reference and second as value. So, any change made in `a` which is passed as reference in callee will be reflected in caller. So, the variable `a`, updated in callee as `(10 + 10) = 20` reflects in caller.

Question 8

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

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

#define MUL(x,y) x*y

int main() {
    int a = 10, b = 5, c, d;

    c = MUL(a, b + 1);
    d = MUL(a + 1, b);

    cout << c << " " << d;

    return 0;
}
```

What will be the output?

a) `60 55`

b) `51 15`

c) `60 15`

d) `51 55`

Answer: b)
Explanation:
`inline` function substitutes all variables used by it before compilation. So, `MUL(a,b+1)` will be substituted as `a*b+1` which is `10*5+1 = 50 + 1` = 51. Similarly, `MUL(a+1,b)` will be substituted as `a+1*b` which is `10+1*5 = 10 + 5 = 15`.

Question 9

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

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

int main() {
    const int *a = new int[2]; // LINE-1

    cout << *a << " " << *(a + 1);

    return 0;
}
```

Modify `LINE-1` such that it will print `5 10`.

a) `const int *a = new int(2){5, 10};`

b) `const int *a = new int[2]{5, 10};`

c) `const int *a = new int[2](5,10);`

d) `const int *a = new int(2)(5,10);`

Answer: b)
Explanation:
`a` is a pointer to a constant integer - eventually to an array of two int as dynamically allocated. It should be initialized during declaration for the desired output. So, `LINE-1` needs to be modified as `const int *a = new int[2]{5, 10};`.

Programming Questions

Question 1

The following program is used to multiply two complex numbers. Fill in the blanks (in `LINE-1`, `LINE-2` and `LINE-3`) so that it will satisfy sample input and output. *Marks: 3*

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

struct Complex {
    int x, y;
};

Complex _____________(Complex &p1, Complex &p2) { // LINE-1
    struct Complex p3 = { 0, 0 };

    p3.x = _____________; // LINE-2
    p3.y = _____________; // LINE-3

    return p3;
}

int main() {
    struct Complex p1, p2;
    cin >> p1.x >> p1.y >> p2.x >> p2.y;

    struct Complex p3 = p1*p2;
    cout << p3.x << " " << p3.y;

    return 0;
}
```

Public 1

```
Input: 1 2 3 4
Output: -5 10
```

Public 2

```
Input: 1 -1 0 1
Output: 1 1
```

Private

```
Input: 1 5 2 10
Output: -48 20
```

Answer:
`LINE-1`: `operator*`
`LINE-2`: `p1.x*p2.x - p1.y*p2.y`
`LINE-3`: `p1.x*p2.y + p1.y*p2.x`
Explanation:
We have to override `'*'` operator to multiply complex numbers. So, `LINE-1` should be filled with header of operator overloading function `operator*`. In `LINE-2` and `LINE-3`, multiplication of complex numbers is done. So, it will be filled up with
`LINE-2`: `p1.x*p2.x - p1.y*p2.y`
`LINE-3`: `p1.x*p2.y + p1.y*p2.x`

Question 2

Consider the following program and fill in the function header `print()` at `LINE-1` such that it matches the given test cases. *Marks: 3*

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

void _____________________ { // LINE-1
    cout << a << " " << b;
}

int main() {
    string p;
    cin >> p;

    if (p == "x" || p == "X")
        print("Hello");
    else
        print("Hello", p);

    return 0;
}
```

Public 1

```
Input: Sir
Output: Hello Sir
```

Public 2

```
Input: x
Output: Hello Anyone
```

Private

```
Input: Student
Output: Hello Student
```

Answer:
`LINE-1`: `print(string a, string b = "Anyone")`
Explanation:
The function header should have two parameters. The second parameter should have default value "Anyone". So, `LINE-1` can be filled with
`print(string a, string b = "Anyone")`

Question 3

Consider the following program and fill in the blanks in `LINE-1` with appropriate function header so that it will take one argument as call by reference and in `LINE-2` for the return statement. Consider the given test cases. *Marks: 3*

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

int Double(__________) { // LINE-1
    return __________;   // LINE-2
}

int main() {
    int x, y;
    cin >> x >> y;

    cout << Double(x + y);

    return 0;
}
```

Public 1

```
Input: 2 4
Output: 12
```

Public 2

```
Input: 5 10
Output: 30
```

Private

```
Input: 2 -2
Output: 0
```

Answer:
`LINE-1`: `const int &n`
`LINE-2`: `2*n` or `(n+n)`
Explanation:
The function call is made with an argument which is a constant expression. As the function is called with call by reference strategy, and actually an expression (`x+y`) is passed, the argument should be constant in nature. So, `LINE-1` should be filled as `const int &n`. The function gives double value as output. So `LINE-2` should be filled as `2*n` or `(n+n)`.

Question 4

Consider the following program and fill in the blanks at LINE-1, and LINE-2. LINE-2 should be filled with dynamic memory allocation code which will allocate memory to the pointer p for three integers. LINE-3 should be filled with memory deletion code. Consider the sample test cases. *Marks: 3*

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

void process(int *p) {
    for (int i = 0; i<3; i++)
        cin >> *(p + i);

    for (int i = 2; i >= 0; i--)
        cout << *(p + i) << " ";
}

int main() {
    int *p;

    p = __________; // LINE-1

    process(p);

    __________;     // LINE-2

    return 0;
}
```

Public 1

```
Input: 1 2 3
Output: 3 2 1
```

Public 2

```
Input: 1 5 9
Output: 9 5 1
```

Private

```
Input: 2 4 6
Output: 6 4 2
```

Answer:

LINE-1: `p = new int[3];`
LINE-2: `delete[] p;`

Explanation:

The integer pointer p is passed to function. So, LINE-1 is filled with `int *p;`. LINE-2 will allocate three integer memory to the pointer p. So, it can be filled as `p = new int[3];`. LINE-3 will be filled as `delete[] p`.