Programming in Modern C++: Assignment Week 2

Total Marks : 25

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

January 26, 2025

Question 1

Consider the following function prototypes of function `add()`. *[MCQ, Marks 2]*

P1: `int add(int n1 = 0, int n2);`

P2: `int add(int = 0, int = 0);`

P3: `double add(double = 0, double d = 0.0);`

P4: `double add(int = 0, double d = 0.0);`

P5: `double add(int n1 = 0, int n2, double n3 = 0);`

Which of the following sets consists of all valid prototypes of function `add()`?

a) {`P1, P3`}

b) {`P2, P3, P4`}

c) {`P1, P2, P3, P4`}

d) {`P2, P3, P5`}

Answer: b)
Explanation:
Default parameters need to be initialized from right to left. That is, once a parameter is defaulted, all parameters to its right must also be defaulted. Also, it is optional to specify the formal parameter in a function prototype - only type suffice.

Question 2

Consider the following function prototypes of function `add()`. *[MSQ, Marks 2]*

P1: `int add(int n1);`

P2: `int add(int = 0, int = 0);`

P3: `double add(int = 0, double d = 0.0);`

P4: `int add(int = 0, double d = 0.0);`

P5: `double add(int n1 = 0, int n2 = 0, double n3 = 0);`

Which of the above pairs of the prototypes is ambiguous and illegal as per the rule of function overloading?

a) `P1, P2`

b) `P2, P3`

c) `P3, P4`

d) `P2, P5`

Answer: d)
Explanation:
Each overloaded function must have a unique parameter list - the number of parameters or the type of the parameters need to be different from others, which is true for option a), b) and d). However, for P3 and P4, the number of parameters and their types are same which is an incorrect overloading.
This question is intentionally made as MSQ

Question 3

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

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

int add(int n1 = 100) { return n1; }
int add(int n1 = 100, int n2 = 100) { return n1 + n2;}
int add(int n1 = 100, double d1 = 100.0){ return n1 + d1; }
int add(int n1 = 100, int n2 = 100, int n3 = 100) { return n1 + n2 + n3; }

int main() {
    int r = add(10, 20);
    cout << r << endl;
    return 0;
}
```

What will be the output/error ?

a) `30`

b) `130`

c) `300`

d) `Compilation Error:  call of overloaded ‘add(int, int)’ is ambiguous`

Answer: d)
Explanation:
The call `add(10, 20)`, can invoke both the following overloads of `add` function:

```
int add(int n1 = 100, int n2 = 100) { ...  }
int add(int n1 = 100, int n2 = 100, int n3 = 100) { ...  }
```

Thus, the call is ambiguous.

Question 4

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

```
#include <iostream>
using namespace std;
#define DOUBLE(x) (2 * x)

inline int TRIPLE(int x){
    return 3 * x;
}
int main(){
    cout << DOUBLE(10 + 10) << " ";
    cout << TRIPLE(10 + 10) << " ";
    return 0;
}
```

What will be the output?

a) `40 60`

b) `30 60`

c) `30 40`

d) `20 30`

Answer: b)
Explanation:
In a macro call, the arguments get substituted blindly, and then evaluated. However, in inline function call, the arguments get evaluated before those are passed to the function.
The call `DOUBLE(10 + 10)` will be expanded as:
`2 * 10 + 10`, which results in `30`.
However, the call to the inline function `TRIPLE(10 + 10) => TRIPLE(20)` results in `60`.

Question 5

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

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

__________________ {    //LINE-1
    return ++i;
}
int main() {
    int x = 10, y = 20;
    int& z = incr(x);
    cout << x << " " << z << " ";
    incr(x) = y;
    cout << x << " " << z;
    return 0;
}
```

Choose the correct option to fill in the blank at `LINE-1` such that the output is `11 11 20 20`.

a) `int incr(int i)`

b) `int incr(int& i)`

c) `int& incr(const int& i)`

d) `int& incr(int& i)`

Answer: d)
Explanation:
Since the increment of the formal parameter `i` is reflected on the actual variable `x`, it must be passed as reference. However, not as constant reference, since the formal parameter is modified within the function.
The statement `int& z = incr(x);` requires the return type to be a reference type. The d) is the correct option.

Question 6

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

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

#define C1 10 + 1
const int C2 = 10 + 1;

int main(){
    int n1, n2;
    n1 = C1 * 100 * C2;
    n2 = C2 * 100 * C1;
    cout << n1 << " " << n2;
    return 0;
}
```

What will be the output?

a) `1110 11001`

b) `1011 1011`

c) `12100 12100`

d) `11001 1110`

Answer: a)
Explanation:
Manifest constant is replaced by the expression it defines. So `C1` will be replaced by `10 + 1`. On the other hand, the value of `C2` is `11`.
Thus, `n1 = C1 * 100 * C2;` is evaluated as follows:
`n1 = C1 * 100 * C2 => 10 + 1 * 100 * 11 => 10 + 100 * 11 => 10 + 1100 => 1110`
And `n1 = C2 * 100 * C1;` is evaluated as follows:
`n1 = C2 * 100 * C1 => 11 * 100 * 10 + 1 => 1100 * 10 + 1 => 11000 + 1 => 11001`.

Question 7

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

```
#include <iostream>
using namespace std;
int main() {
    int i = 10;
    int &r = i;
    r += i += ++r;
    cout << i << " " << r;
    return 0;
}
```

What will be the output?

a) `21 32`

b) `32 32`

c) `21 44`

d) `44 44`

Answer: d)
Explanation:
Since `r` is a reference to `i`, they are alias. The statement `r += i += ++r;` is evaluated as follows:

```
r += i += (++r);   //r = 11, i = 11
r += (i += 11);    //r = 11, i = 11
r += 22;           //r = 22, i = 22
r = 44;            //r = 44, i = 44
```

Question 8

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

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

int main(){
    const char c = 'X';
    ____________ = &c;    //LINE-1
    cout << *cp++;
    return 0;
}
```

Identify the correct statement/s to fill in the blank at `LINE-1` such that the output is `X`.

a) `const char* cp`

b) `char* const cp`

c) `char const* cp`

d) `const char * const cp`

Answer: a), c)
Explanation:
Since `c` is a character constant, `cp` must be a pointer to a constant pointee. Thus, at `LINE-1` assignment is valid for options a), c), and d). However, in statement `cout << *cp++`, the operation `cp++` is not possible for option d). So, the correct options are a) and c).

Question 9

Consider the following statement in C. *[MSQ, Marks 2]*

```
int *ip = (int *)malloc(sizeof(int)* 5);
```

Among the given options below, identify the equivalent statement/s (perform the same task) in C++.

a) `int *ip = new int(5);`

b) `int *ip = new int[5];`

c) `int *ip = (int *)operator new(sizeof(int) * 5);`

d) `int *ip = (int *)operator new(sizeof(int))[5];`

Answer: b), c)
Explanation:
Option b) and c) are the equivalent statements (allocate memory for 5 integers) using `operator new()` and `operator new[]`, respectively.
Option a) allocates memory for 1 integer and initializes with 5. Hence, it is wrong.
Option d) is a wrong statement.

Programming Questions

Question 1

Consider the program below which defines a type `point` and overloads operator `+` such that the `x` and `y` coordinates of a given `point` can be added with an integer.
Complete the program with the following instructions.

- Fill in the blank at `LINE-1` with the appropriate header of the function to overload operator `+`.
- Fill in the blank at `LINE-2` to declare a new `point`.
- Fill in the blank at `LINE-3` to return the new `point`.

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

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

struct point {
    int x, y;
};

_________________________ {    //LINE-1
    ____________________;      //LINE-2
    new_pt.x = pt.x + t;
    new_pt.y = pt.y + t;
    ____________________;      //LINE-3
}

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    point p = {a, b};
    int t = c;
    point np = p + t;
    cout << "(" << np.x << ", " << np.y << ")";
    return 0;
}
```

Public 1

```
Input: 10 20 -5
Output: (5, 15)
```

Public 2

```
Input: 100 100 10
Output: (110, 110)
```

Private 3

```
Input: -10 20 5
Output: (-5, 25)
```

Answer:

```
LINE-1:  point operator+(point pt, int t)
LINE-2:  point new_pt
LINE-3:  return new_pt
```

Explanation:

At `LINE-1`, the header of the operator overloading function shall be defined as:

```
point operator+(point pt, int t)
```

At `LINE-2`, the new `point` object can be declared as:

```
point new_pt;
```

At `LINE-3`, the return statement shall be:

```
return new_pt;
```

Question 2

Consider the following program. Fill in the blanks at `LINE-1` and `LINE-2` with appropriate headers of overloaded function `product`. The program must satisfy the sample input and output. *Marks: 3*

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

------------------------------ {
    return a * b * c;
}

---------------------------- {
    return a * b;
}

int main() {
    int i1, i2, i3;
    double d1, d2;
    cin >> i1 >> i2 >> i3 >> d1 >> d2;
    cout << product(i1) << ", ";
    cout << product(i1, i2) << ", ";
    cout << product(i1, i2, i3) << ", ";
    cout << product(d1, d2);
    return 0;
}
```

Public 1

```
Input: 10 20 30 10.5 5.67
Output: 10, 200, 6000, 59.535
```

Public 2

```
Input: -10 20 50 3.14 10
Output: -10, -200, -10000, 31.4
```

Private

```
Input: -2 3 -5 3.1 5.6
Output: -2, -6, 30, 17.36
```

Answer:

```
LINE-1:  int product(int a, int b = 1, int c = 1)
LINE-2:  double product(double a, double b)
```

Explanation:
The function header that allows all the following calls:
`product(i1), product(i1, i2), product(i1, i2, i3)` is
`int product(int a, int b = 1, int c = 1)`.
The function header that allows function call `product(d1, d2)` is
`double product(double a, double b)`

Question 3

Consider the following program. Fill in the blank at `LINE-1` with an appropriate function header for the function `update_and_sum( )` such that it satisfies the sample input and output. *Marks: 3*

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

___________________________________ {    //LINE-1
    a *= 10;
    b *= 10;
    c = a + b;
    cout << a << ", " << b << ", " << c << endl;
}
int main() {
    int i1, i2, i3 = 0;
    cin >> i1 >> i2;
    update_and_sum(i1, i2, i3);
    cout << i1 << ", " << i2 << ", " << i3 << endl;
    return 0;
}
```

Public 1

```
Input: 10 20
Output:
100, 200, 300
10, 20, 300
```

Public 2

```
Input: -10 300
Output: -100, 3000, 2900
-10, 300, 2900
```

Private

```
Input: -20 -10
Output:
-200, -100, -300
-20, -10, -300
```

Answer:

```
void update_and_sum(int a,int b,int &c)
```

Explanation:

Since the updates performed inside the function on the first two arguments are not reflected on the actual arguments, they must be passed as pass-by-value mechanism. However, for the third argument the update performed inside the function is reflected on the actual argument, it must be passed as pass-by-reference.